UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19|20

Trabajo fin de máster

Estudio de una Red Neuronal Recurrente Dual aplicada a la generación de secuencias

Christian Oliva Moya

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

Universidad Autónoma de Madrid

Escuela Politécnica Superior

Proyecto para la obtención del título de
Máster en Investigación e Innovación en
Inteligencia Computacional y Sistemas
Interactivos (I2-ICSI)
por la Universidad Autónoma de Madrid

Tutor del trabajo fin de máster:
Luis F. Lago Fernández

Estudio de una Red Neuronal Recurrente Dual aplicada a la generación de secuencias

Christian Oliva Moya

Christian Oliva Moya
Estudio de una Red Neuronal Recurrente Dual aplicada a la generación de secuencias

Christian Oliva Moya
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi novia Rocío y a mi familia

Resumen

Se ha realizado un estudio de la equivalencia entre las redes neuronales recurrentes y los autómatas finitos deterministas con el fin de proporcionar un mecanismo de interpretación para este tipo de redes profundas. En primer lugar, se presenta un análisis empírico de la estabilidad y la capacidad de generalización de una red recurrente cuando se inyecta ruido Gaussiano a las neuronas de la capa oculta justo antes de aplicar la función de activación. Además, se demuestra que las redes entrenadas en estas condiciones con lenguajes regulares se comportan como los autómatas finitos equivalentes.

En segundo lugar, se desarrolla una nueva arquitectura de red recurrente, la red Dual, que mejora la generalización y la interpretabilidad utilizando dos rutas diferentes en el procesamiento de la información. Por un lado, una capa recurrente se encarga de procesar las dependencias temporales en las secuencias de entrada. Por otro lado, una capa densa combina la salida de la capa recurrente con la información presente en la entrada para proporcionar la salida final de la red. En conjunto, la nueva arquitectura Dual admite una interpretación como una máquina de Mealy. Los resultados obtenidos tanto en problemas sintéticos como en problemas reales muestran, por un lado, que el reparto de la carga de procesamiento de la información simplifica la complejidad de la capa recurrente y, por otro lado, que la inyección de ruido mejora considerablemente la capacidad de generalización de la red y su posible interpretabilidad, consiguiendo mejorar ligeramente el resultado de una LSTM estándar entrenada con los mismos problemas.

Palabras clave

Redes neuronales recurrentes, Autómatas finitos deterministas, Interpretabilidad, Aprendizaje automático interpretable

Abstract

A study of the equivalence between recurrent neural networks and deterministic finite automata has been carried out. First, an empirical analysis of the stability and generalization of a recurrent neural network when noise is applied to the recurrent layer is presented, and it is shown that networks trained on regular languages under these conditions behave as the equivalent Deterministic Finite Automaton. Second, a new recurrent neural network architecture is developed, the Dual RNN, which improves generalization and interpretability by using two different information processing paths. On the one hand, a recurrent layer deals with the temporal dependencies present in the network's input. On the other hand, a dense feedforward layer combines the output of the recurrent layer with the input to obtain the network's output. This new Dual architecture admits an interpretation as a Mealy machine. The results obtained with both synthetic and real problems show that the division of the processing simplifies the complexity of the recurrent layer. Also, the noise injection considerably improves both the generalization capacity and its interpretability, beating a standard LSTM trained on the same problems.

Keywords

Recurrent Neural Networks, Deterministic Finite Automata, Interpretability, Interpretable Machine Learning

Índice

Listas

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

La capacidad de entendimiento, comprensión y resolución de los problemas a los que uno se enfrenta día a día es un ejercicio que, desde los orígenes de la informática y la ciencia de la computación, se ha planteado como objetivo para definir la disciplina de la inteligencia artificial, que podría definirse como la capacidad de un sistema para interpretar y aprender de un conjunto de datos externos. Hoy en día, este paradigma multidisciplinar abarca una amplia gama de soluciones en las áreas de desarrollo de software que requieren la comprensión y modelado, como son el aprendizaje automático, la minería de datos, la robótica o las finanzas, entre muchas otras.

El aprendizaje automático, que es una de las disciplinas que mayor auge está teniendo en este siglo de la digitalización, tiene como objetivo crear modelos abstractos capaces de extraer información de un conjunto de datos de acuerdo a sus características. Aplicado a numerosos problemas de diferente envergadura, tales como el aprendizaje de lenguajes formales [1–7], el procesamiento de imágenes [8,9], o el procesamiento del lenguaje natural [10–12], el aprendizaje automático se ha vuelto esencial en la ciencia de la computación y el desarrollo de software de nuestros días. Concretamente, las redes neuronales artificiales, que no son más que uno de los múltiples modelos de esta disciplina, encabezan la carrera de la investigación con un amplio abanico de posibilidades. Desde los orígenes de esta alternativa en los años 40 con el perceptrón hasta las colosales redes profundas de hoy en día, buscando imitar la capacidad cognitiva y las conexiones sinápticas del cerebro humano, estas redes han sufrido épocas de decadencia y esplendor que, gracias al *Big Data* y la capacidad de computación de la que se dispone hoy día, suponen una revolución tecnológica sin precedentes. Las redes neuronales recurrentes y el aprendizaje profundo definen el camino hacia el futuro, donde, cada vez más, la inteligencia artificial cobra un mayor protagonismo.

Una red neuronal recurrente (RNN) es un tipo de red neuronal artificial cuya arquitectura está especialmente diseñada para modelar datos con una estructura temporal, ya que introduce una conexión recurrente en la que el estado interno de la red depende tanto de la entrada actual como de su estado previo. Esta sencilla, pero interesante idea, desarrollada en los años 90 [13], ha resultado ser de vital importancia para procesar datos secuenciales con una relación temporal, como es el reconocimiento de voz [14], el procesamiento de lenguaje natural [10–12] o la composición de música [15], entre muchos otros.

Es posible demostrar que una RNN es un modelo Turing-completo [16], por lo que, desde sus comienzos, se ha buscado la equivalencia entre estas redes y los autómatas que reconocen los lenguajes formales. En particular, si una red es entrenada para identificar un lenguaje regular, esta debería ser equivalente al autómata finito determinista que lo identifica [17]. Desde el inicio de esta idea en los 90 [4], son muchos los autores que han estudiado el paralelismo entre ambos modelos y la habilidad de las RNNs para aprender lenguajes formales [18]. Además, en los últimos años, con la entrada en escena de las redes neuronales con memoria aumentada [19–21], que buscan aproximar este tipo de modelos al concepto abstracto de la máquina de Turing utilizando algún mecanismo de memoria, se ha dado un paso más en esta búsqueda.

En esta línea de investigación, en este trabajo de fin de máster se pretende explorar la capacidad de interpretación de las RNNs buscando la equivalencia con los autómatas finitos deterministas. Con este objetivo, se ha profundizado en dos áreas ligeramente diferentes: en primer lugar, se ha explorado el modelo de la red de Elman [13] con ciertas modificaciones [7] con el objetivo de analizar la estabilidad que el ruido inyectado en la función de activación provoca en el modelo. Si la red encuentra una estabilidad en su espacio interno de estados, es posible afirmar que la red es equivalente al autómata finito correspondiente y, por tanto, queda clara la interpretación directa del modelo. Por otro lado, en este trabajo se ha desarrollado la arquitectura de la red recurrente Dual, una RNN basada en las modificaciones de la red de Elman que, de forma ingeniosa, separa el procesamiento de la información temporal de una secuencia del resto del proceso. Esta estrategia parece beneficiar no solo el aprendizaje del modelo, sino también su interpretabilidad y su simplicidad. Desde un punto de vista abstracto, esta arquitectura provoca que la respuesta del modelo dependa tanto de su estado interno como de la entrada en un instante de tiempo determinado, simulando el comportamiento lógico de una máquina de Mealy, por lo que su interpretabilidad vuelve a ser clara. Siguiendo estas dos aproximaciones, los objetivos de este trabajo son los siguientes:

- Estudiar y analizar el comportamiento del ruido y la estabilidad que este proporciona al modelo de la red de Elman modificada, buscando una equivalencia con un autómata finito determinista.
- Estudiar y analizar en profundidad la red Dual y su interpretabilidad en diferentes problemas, buscando la equivalencia con una máquina de Mealy. Concretamente se probará con los problemas siguientes:
    - Lenguajes regulares sencillos
    - Suma de dos sumandos en diferentes bases
    - Generación de expresiones algebraicas con profundidad de paréntesis máxima fija
    - Procesamiento de lenguaje natural: generación de texto y análisis de sentimiento
- Comparar los resultados obtenidos de ambos modelos con el modelo estándar LSTM.

Este documento está dividido en cinco capítulos además de esta introdución: en el capítulo 2 se definen los conceptos básicos que deben conocerse antes de comenzar con el contenido del trabajo. En el capítulo 3 se presenta el estado de la cuestión sobre el campo de investigación del aprendizaje automático y el procesamiento del lenguaje natural. En el capítulo 4 se describen los conjuntos de datos utilizados y el diseño de las redes desarrolladas. Por último, antes de dar paso a las conclusiones, en el capítulo 5 se presenta el análisis y los resultados obtenidos a lo largo del desarrollo del proyecto. Adicionalmente, se incluyen apéndices con la implementación de la red de Elman modificada en *Keras* [22] (apéndice A) y, por otro lado, algunos resultados que completan los experimentos (apéndices B, C y D).

El transcurso de este proyecto ha brindado la posibilidad de enviar una contribución a una conferencia internacional [23] con los resultados de esta investigación, concretamente aquellos relacionados con la red Dual (ver sección 5.4). Además, se encuentran en fase de desarrollo otras dos contribuciones [24, 25] que detallan los resultados obtenidos con la estabilidad (sección 5.1) y el diseño de la red dual (ver secciones 4.2.2 y 5.2).

2

Conceptos básicos

En este apartado se introducen una serie de conceptos básicos que el lector debe conocer antes de comenzar con el contenido de este documento. Con este objetivo, este capítulo se divide en las siguientes secciones: en la sección 2.1 se describen las clases de lenguajes formales que van a ser referenciadas a lo largo del texto: los lenguajes regulares (sección 2.1.1), los lenguajes independientes de contexto (sección 2.1.2) y los lenguajes recursivamente enumerables (sección 2.1.3). En la sección 2.2 se describe conceptualmente el aprendizaje automático y se introducen las redes neuronales artificiales (sección 2.2.1). Por último, en la sección 2.3 se detalla el concepto de aprendizaje profundo y se definen tanto las redes neuronales recurrentes (sección 2.3.1) como las redes neuronales con memoria aumentada (sección 2.3.2), dos arquitecturas muy referenciadas a lo largo del texto.

2.1. Lenguajes formales

Un lenguaje formal se define como un conjunto de cadenas generadas a partir de una gramática formalmente especificada. Una gramática formal [26] es una cuádrupla $G = (N, \Sigma, P, A)$ donde N es un conjunto de símbolos no terminales, es decir, se pueden sustituir con una regla de producción y se suelen representar en mayúsculas; Σ es el alfabeto, símbolos finales que no pueden ser sustituidos que se suelen representar en minúsculas; P es un conjunto de producciones o reglas de transformación; y $A \in N$ es el axioma, es decir, el símbolo raíz o inicial. Las reglas de producción P determinan cómo transformar el axioma A en una cadena del lenguaje formada exclusivamente por símbolos terminales.

Chomsky [27] define una serie de tipos de gramática, donde sus diferencias se encuentran en las restricciones que tienen sus producciones. Cada uno de los tipos de gramática es equivalente a un autómata que lo reconoce, es decir, dicho modelo es capaz de identificar cadenas pertenecientes al lenguaje generado por la gramática. Los tipos de gramática de Chomsky se introducen a continuación:

- **Tipo-3**: Este tipo de gramáticas genera los lenguajes regulares, que son el conjunto de lenguajes formales más restrictivos de acuerdo a la jerarquía de Chomsky [27]. Esta clase de lenguajes puede ser identificada por un autómata finito (sección 2.1.1). En estos lenguajes, el lado izquierdo de una producción debe contener un símbolo no terminal, mientras que el

lado derecho puede contener símbolos terminales y como mucho un símbolo no terminal. Si los símbolos terminales aparecen siempre al inicio se denomina gramática regular lineal izquierda, mientras que si los símbolos terminales aparecen siempre al final se denomina gramática regular lineal derecha.

- **Tipo-2**: Genera los lenguajes independientes de contexto. Este tipo de lenguajes son reconocidos por los autómatas a pila (sección 2.1.2). En estos lenguajes se suprime la restricción de linealidad, es decir, el lado derecho de una producción puede contener símbolos terminales y no terminales sin ninguna condición adicional.
- **Tipo-1**: Este tipo de gramáticas genera los lenguajes dependientes de contexto. Este tipo de gramáticas componen todos los lenguajes que pueden ser reconocidos por un autómata linealmente acotado [26], una forma restrictiva de una máquina de Turing. Este tipo de lenguajes no va a ser estudiado en este documento.
- **Tipo-0**: Incluye todas las gramáticas formales. Generan exactamente todos los lenguajes que pueden ser reconocidos por una máquina de Turing (sección 2.1.3). Estos lenguajes se conocen también como lenguajes recursivamente enumerables.

2.1.1. Lenguajes regulares: autómatas Finitos

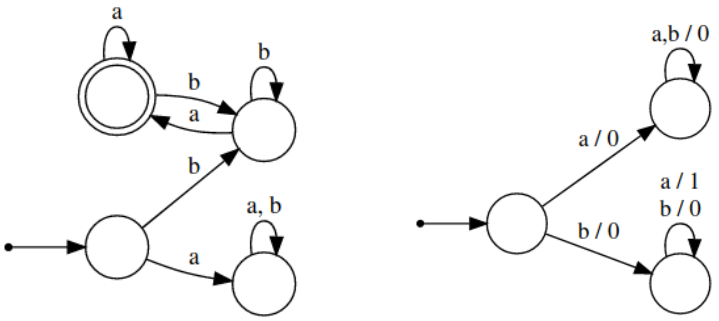
Los lenguajes regulares son el conjunto de lenguajes formales más restrictivos según la jerarquía de Chomsky [27] y se describen mediante las gramáticas regulares, o de tipo-3. Estas gramáticas o bien son lineales por la derecha (los símbolos terminales aparecen siempre al final de la parte derecha de la producción) o bien son lineales por la izquierda (los símbolos terminales aparecen al inicio de la parte derecha de la producción). El lado izquierdo de una producción debe contener exclusivamente un símbolo no terminal. Por otro lado, las expresiones regulares creadas por Stephen Kleene [28] son una herramienta muy común para describir formalmente este tipo de lenguajes. Estas expresiones están formadas por una secuencia de caracteres que constituyen el patrón de todas las cadenas que pertenecen a un determinado lenguaje regular. Se muestra un ejemplo en la tabla 2.1, donde se introduce el lenguaje que contiene todas las cadenas que comienzan con el símbolo b y terminan con el símbolo a, lenguaje denominado de aquí en adelante como BxA.

Gramática regular	Expresión regular
A $\rightarrow$ bB	b(a+b)*a
B $\rightarrow$ bB | aB | a	

Tabla 2.1: Descripción de un lenguaje regular con alfabeto $\Sigma = \{a, b\}$ y su expresión regular equivalente. El símbolo | se utiliza como operador booleano OR en la gramática, mientras que en la expresión regular equivalente, este operador se representa con el símbolo $+$ y, por último, el símbolo $*$ representa 0 o más repeticiones del símbolo o grupo al que aplica.

Las cadenas pertenecientes a un lenguaje regular pueden ser reproducidas a través de un autómata finito. Este hecho implica que existe una equivalencia entre una gramática regular, su expresión regular correspondiente y el autómata finito que la identifica [29]. Por definición, un autómata finito es una quíntupla $M = (Q, \Sigma, \delta, q_0, F)$ donde Q es el conjunto de estados del autómata, Σ es el alfabeto, $\delta : Q \times \Sigma \rightarrow Q$ es la función de transición, $q_0 \in Q$ es el estado inicial y $F \subseteq Q$ es el conjunto de estados de aceptación de cadena (estados finales). El funcionamiento de los autómatas se basa en la función de transición δ, que dado un estado q_{actual} y un símbolo del alfabeto, devuelve un estado $q_{siguiente}$ al que se transita con dicho símbolo. Si después de procesar una cadena completa $w \in \Sigma^*$ [1] el estado interno del autómata es un estado final $q_f \in F$, entonces la cadena w pertenece al lenguaje (cumple las producciones de la gramática formal).

La definición formal de un autómata finito determinista es equivalente a la descripción una máquina de Moore [30], una máquina de estados que determina su salida solamente en base a su estado actual, es decir, la aceptación está asociada al estado y solamente cambia cuando cambia el estado al llegar el siguiente símbolo. Se muestra la máquina de Moore que resuelve el problema BxA en la figura 2.1(a). En este diagrama se define completamente la quíntupla del autómata: cada círculo identifica cada uno de los estados de Q, las flechas entre los estados y los símbolos sobre ellas representan las transiciones δ y el alfabeto Σ, el estado q_0 es el que tiene una flecha inicial y los estados de aceptación F son aquellos con doble círculo. Sin embargo, existe otro tipo de máquinas de estado equivalente: la máquina de Mealy [31], donde la salida se determina a partir del estado actual y la entrada. Esta máquina es capaz de generar salidas diferentes en un mismo estado. De esta forma, es común que la máquina de Mealy sea más sencilla que su máquina de Moore equivalente, tal y como se muestra en la figura 2.1(b). Nótese que en una máquina de Mealy no hay estados finales: la salida, que puede ser 0 ó 1, donde 0 implica rechazar y 1 aceptar, viene dada por el estado y el símbolo de entrada del modelo.



(a) Máquina de Moore (b) Máquina de Mealy

Figura 2.1: Diagrama de estados de Moore y Mealy equivalentes al lenguaje regular BxA

[1] La terminología cierre estrella (Σ^*) representa la aparición de 0 o más símbolos concatenados del conjunto al que aplica. En este caso concreto representa cualquier posible cadena que se puede generar con los símbolos del alfabeto. Este lenguaje (Σ^*) se denomina lenguaje universal sobre el alfabeto.

2.1.2. Lenguajes independientes de contexto: autómatas a pila

Los lenguajes independientes de contexto son el conjunto de lenguajes formales descritos por las gramáticas de tipo-2 descritas por Chomsky [27]. En estos lenguajes se suprime la restricción de linealidad de los lenguajes regulares, es decir, mientras que el lado izquierdo de una producción mantiene un símbolo no terminal de la misma forma que los lenguajes regulares, el lado derecho puede combinar símbolos terminales y no terminales sin ninguna condición adicional: $V \rightarrow w$ donde $V \in N$ y $w \in (N \cup \Sigma)^*$. Se muestra en la tabla 2.2 un ejemplo de lenguaje independiente de contexto: el lenguaje de todas las expresiones algebraicas, denominado $expAlg$.

Gramática independiente de contexto
S $\rightarrow$ S O T | T
T $\rightarrow N$ | (S)
O $\rightarrow + \mid - \mid * \mid /$

Tabla 2.2: Ejemplo de la descripción del lenguaje independiente de contexto $expAlg$. Esta gramática tiene como alfabeto $\Sigma = \{(,), +, -, *, /, N\}$, donde N representa cualquier número. Se utiliza el símbolo | como operador booleano OR.

Continuando con el ejemplo de las expresiones algebraicas, la no linealidad de la producción T $\rightarrow$ (S) provoca que, si se intenta generar el autómata finito que identifica el problema, sea necesario un autómata con infinito número de estados, ya que es necesario recordar la profundidad de apertura y cierre de paréntesis. Cuando se abre un paréntesis, la profundidad aumenta y, en el caso de cierre, disminuye. Por tanto, siguiendo con el ejemplo, una expresión algebraica estaría bien formada cuando cumpla ciertas reglas que pueden definirse como un lenguaje regular y, además, cumpla que se abran y cierren todos los paréntesis correctamente, como es el caso de la producción $T \rightarrow (S)$. La figura 2.2 muestra este concepto teórico.

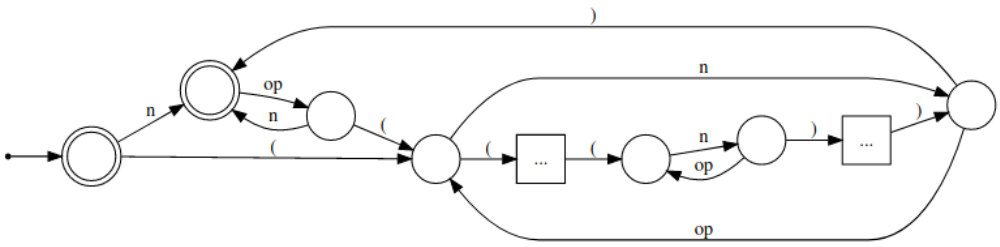


Figura 2.2: Concepto de un autómata finito determinista con un número infinito de estados que identificaría el problema $expAlg$, donde los estados *cuadrados* denominados "..." representan esa recursión infinita de *profundidad*.

Los lenguajes independientes de contexto se identifican con un autómata a pila. Por definición, un autómata a pila se define como una séptula $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, donde Q es un conjunto finito de estados, Σ es el alfabeto de entrada, Γ es el alfabeto de la pila, $\delta : Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma^*)$ es la función de transición, $q_0 \in Q$ es el estado inicial, $Z \in \Gamma$ es el símbolo inicial de la pila y $F \subseteq Q$ es el conjunto de estados de aceptación (estados finales). La interpretación de la función de transición es la siguiente: Cuando el estado del autómata es $q \in Q$, el símbolo es $a \in \Sigma$ y en la cima de la pila el símbolo es $b \in \Gamma$, se transita al estado $q_{siguiente}$ correspondiente y hay tres posibilidades con la pila: (1) extraer de la cima de la pila un símbolo, (2) añadir a la cima uno o más símbolos o (3) dejar la pila intacta [2]. Se muestra en la figura 2.3 el autómata a pila que identifica el problema $expAlg$.

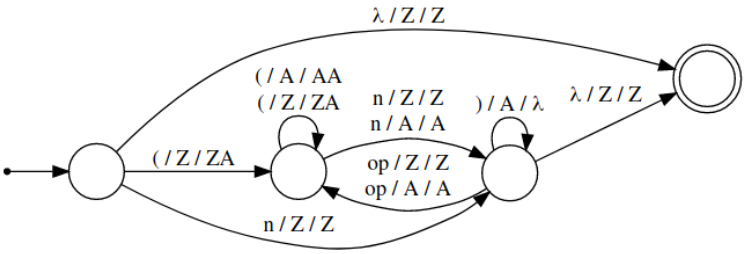


Figura 2.3: Autómata a pila que identifica el problema $expAlg$. En este autómata, el alfabeto es $\Sigma = \{n, op, (,)\}$, donde n es cualquier número y op es cualquier operador; y $\Gamma = \{Z, A\}$, donde Z es el símbolo inicial de la pila y A es un símbolo auxiliar. El símbolo λ representa la cadena vacía.

Las transiciones, entonces, se leerían de la siguiente manera: $n/Z/Z$ representa que aparece un número n en la entrada y hay una Z en la cima de la pila, por tanto dejo la pila como está, Z, y transito al estado correspondiente. Otro ejemplo sería el siguiente: $(/A/AA$ representa que aparece un símbolo $($ en la entrada y hay una A en la cima de la pila, por tanto añado otra A a la pila (AA) y transito al estado correspondiente. Como se puede ver, en este caso la pila actúa como un contador de paréntesis de apertura.

Para concluir, es un hecho que las gramáticas independientes de contexto permiten desarrollar la mayoría de los lenguajes de programación, ya que su sintaxis se puede describir con estos lenguajes. Además, la sintaxis de cualquier lenguaje natural se puede describir como un lenguaje independiente de contexto. De hecho, podría decirse que cualquier lenguaje natural es un lenguaje independiente de contexto con profundidad limitada [32] (en este caso, la profundidad se encuentra en la anidación de subordinación). Usualmente se tiende a hablar con oraciones cortas o con pocas subordinadas anidadas, por lo que, en cualquier caso práctico, se podría interpretar como un lenguaje regular extremadamente complejo. Esta idea se utilizará más adelante en algunos experimentos de este documento.

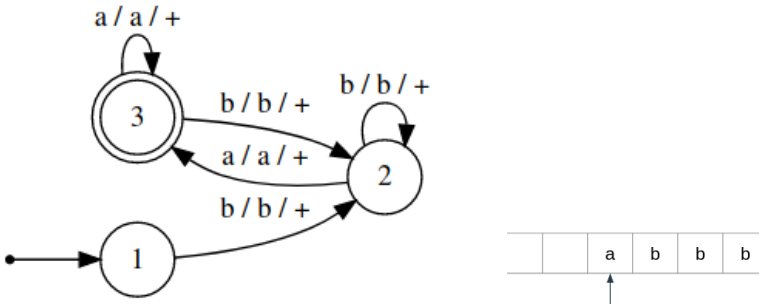
[2] La mecánica exacta del autómata a pila consiste en extraer siempre un símbolo, procesarlo e introducir un elemento de Γ^*.

2.1.3. Lenguajes recursivamente enumerables: máquinas de Turing

Los lenguajes recursivamente enumerables incluyen todas las gramáticas formales. También se conocen como lenguajes Turing-computables, precisamente porque son aquellos lenguajes que pueden ser identificados por una máquina de Turing [33]. Todos los lenguajes regulares, independientes de contexto, dependientes de contexto y recursivos son recursivamente enumerables.

Una máquina de Turing es un modelo que manipula símbolos sobre una cinta infinita de acuerdo a una tabla de reglas. De esta forma, es posible simular la lógica de cualquier algoritmo o problema computable. Formalmente, una máquina de Turing es una séptupla $M = (Q, \Sigma, \Gamma, q_0, b, F, \delta)$, donde Q es un conjunto finito de estados, $\Sigma \subset \Gamma$ es el alfabeto de la entrada, Γ es el alfabeto de la cinta, $q_0 \in Q$ es el estado inicial, $b \in \Gamma$ es el símbolo denominado *blanco*, único símbolo que puede aparecer un número infinito de veces en la cinta, $F \subseteq Q$ es el conjunto de estados finales, y $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ es una función de transición, donde L es un movimiento a la izquierda en la cinta y R es un movimiento a la derecha en la cinta.

Una definición informal describiría una máquina de Turing como una máquina de estados que contiene una cinta de longitud infinita por ambos lados, dividida en celdas. Cada celda contiene un símbolo del alfabeto Γ, incluido el símbolo b (blanco). Además, contiene un cabezal que puede leer y escribir símbolos en la cinta y desplazarse una única celda a la izquierda o a la derecha en cada instante de tiempo. Dado el estado actual y el símbolo leído sobre la cinta, se transita a un nuevo estado, se escribe un nuevo símbolo en la cinta y se desplaza el cabezal. Aunque pueda parecer complejo, se muestra en la figura 2.4 un ejemplo de una máquina de Turing que resuelve el problema *BxA*.



(a) Autómata de la máquina de Turing (b) Cinta de la máquina de Turing

Figura 2.4: Ejemplo sencillo de una máquina de Turing que resuelve el problema *BxA*. Como se puede observar, la nomenclatura se asemeja a un autómata a pila. La cinta de longitud infinita tiene blancos tanto a la derecha como a la izquierda. El cabezal de la cinta está apuntando al símbolo a.

Con este diagrama se define completamente la máquina de Turing. Si la máquina estuviese en la situación de la figura y se encontrase en el estado 2, debería seguir la transición $a/a/+$, que representa lo siguiente: en la cinta hay un símbolo a, entonces lo sustituyo por un símbolo a, desplazo el cabezal a la derecha (+) y transito al estado 3. Un detalle importante a tener en cuenta es que, a lo largo de todo el documento, cuando se haga referencia a una máquina de Turing, se está referenciando una máquina de Turing determinista: un modelo real que describe de forma lógica el funcionamiento de una CPU y puede identificar cualquier gramática formal.

Aunque se aleja de lo que nos concierne en este trabajo, los lenguajes recursivamente enumerables y las máquinas de Turing tienen un papel fundamental en la teoría de complejidad y ciencias de la computación. En estas áreas de investigación son fundamentales los conceptos ideales de máquina de Turing no determinista, es decir, máquinas de Turing con distintas alternativas simultáneas en un estado concreto; y las clases de complejidad: los problemas *P*, *NP* y *NP-completo* [26].

2.2. Aprendizaje automático

El aprendizaje automático [34, 35] es un campo de la ciencia de la computación y la inteligencia artificial [36] cuyo objetivo es crear modelos capaces de extraer información de un conjunto de datos de acuerdo a sus características o atributos. Aquellos algoritmos que requieren que los datos hayan sido previamente etiquetados para poder clasificarlos según dichos atributos forman parte del aprendizaje supervisado. Si, además, los modelos generados dependen de ciertos parámetros que deben ser ajustados durante el aprendizaje, se está haciendo referencia al aprendizaje paramétrico. Aunque existen otras estrategias de aprendizaje, en este proyecto se trata, exclusivamente, el aprendizaje supervisado y paramétrico utilizando redes neuronales.

Es un hábito común validar el modelo elegido utilizando una división del conjunto total de los datos en dos subconjuntos disjuntos: el conjunto de entrenamiento y el conjunto de test. Se dice que un modelo tiene la capacidad de generalizar cuando obtiene un resultado similar al entrenamiento cuando se aplica el test. En caso contrario se estaría produciendo un sobreajuste a los datos de entrenamiento. El proceso del aprendizaje automático consta de cuatro fases bien diferenciadas:

- La recolección de los datos para analizarlos y clasificarlos.
- La selección de los atributos y el modelo a utilizar.
- El entrenamiento del modelo, es decir, el ajuste paramétrico utilizando los datos del conjunto de entrenamiento.
- La validación del modelo, de la que se pueden obtener dos resultados: o bien el entrenamiento es correcto y el modelo es el adecuado, o bien el resultado no es el adecuado y se precisa modificar alguno de los puntos anteriores.

2.2.1. Redes neuronales artificiales

Una red neuronal artificial es un modelo de aprendizaje automático basado en un paradigma inspirado en el funcionamiento biológico de las neuronas [37]. Una neurona artificial se define como una unidad lógica que simula un punto de conexión en una red neuronal artificial. La neurona recibe información de otras neuronas de la red de forma controlada por los parámetros de la misma y tiene la capacidad de propagar la información a aquellas neuronas con las que tiene conexión. La siguiente ecuación describe la activación de una neurona y con N conexiones:

$$y = f(\sum_{i=1}^{N} w_i x_i + b) \tag{2.1}$$

donde x_i representa cada uno de los estímulos de entrada, w_i es el peso de la conexión entre x_i y la neurona, b es el sesgo (bias) de activación de la neurona y f es una función de activación no lineal. Nótese que el argumento de la función f es completamente lineal, mientras que la función introduce un alto grado de no-linealidad. Entrenar una red neuronal consiste en ajustar los parámetros, es decir, los pesos de las conexiones entre las neuronas, para que el modelo aproxime su respuesta a la salida esperada. Existen diferentes arquitecturas cuyo objetivo es definir las conexiones entre las neuronas, pero la más utilizada es la distribución por capas en una red con propagación hacia adelante (*feedforward*), donde cada capa está completamente conectada con la siguiente, tal y como se muestra en la figura 2.5.

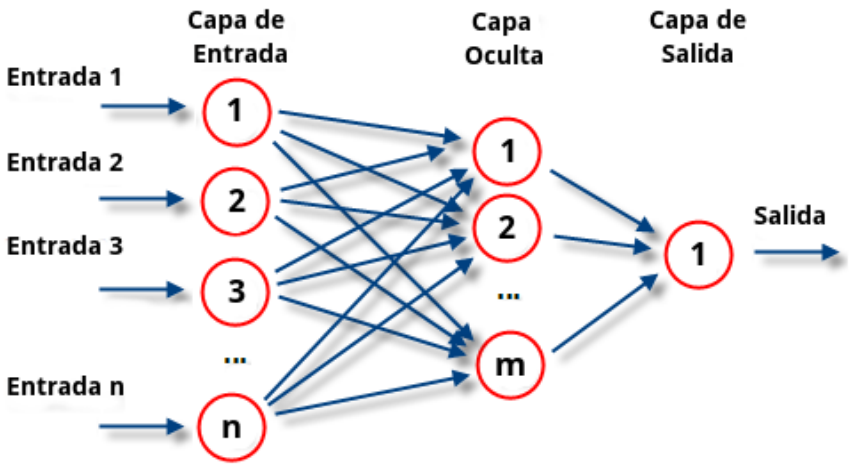


Figura 2.5: Red neuronal multicapa. Nótese que, aunque el diagrama tenga una única neurona en la capa de salida, es posible diseñar redes multicapa con las neuronas de salida necesarias. [Imagen creada por *Gengiskanhg* para Wikipedia. - https://commons.wikimedia.org/wiki/File:RedNeuronalArtificial.png].

En este tipo de redes, las neuronas se agrupan en una capa de entrada (*input*), una o varias capas ocultas (*hidden*) y una capa de salida (*output*), donde cada capa puede estar compuesta por un número diferente de neuronas. La propagación hacia adelante se observa en el hecho de que cada capa solamente establece conexiones con su consecutiva. El proceso de ajuste de los parámetros del modelo se realiza mediante un descenso por gradiente, cuyas ecuaciones dan lugar a la regla de retropropagación [38].

2.3. Aprendizaje profundo

El aprendizaje profundo [39, 40] es un subconjunto de las redes neuronales artificiales que intentan modelar arquitecturas computacionales con múltiples transformaciones no lineales o iterativas. En otras palabras, son aquellas redes con un elevado número de capas o que contienen algún tipo de conexión recurrente que permita transmitir la información hacia atrás en el tiempo durante el entrenamiento. Hoy en día, estos modelos están teniendo un gran protagonismo en problemas como la clasificación de imágenes [8, 9], la generación de música [15], el reconocimiento de voz [14] o el procesamiento de lenguaje natural [10–12].

2.3.1. Redes neuronales recurrentes

Una red neuronal recurrente (RNN) es un tipo de red neuronal multicapa donde el estado interno de la red en un instante de tiempo h_t depende tanto de la entrada de la red x_t como de su estado anterior h_{t-1}. Este efecto se consigue gracias a una conexión recurrente, donde la salida de la capa oculta conecta tanto con la capa siguiente como consigo misma. Gracias a esta conexión, conceptualmente se puede afirmar que son un tipo de redes profundas con profundidad *infinita*. Se describe a continuación la arquitectura clásica de la red neuronal de Elman [13], cuyas funciones de activación se describen con las siguientes ecuaciones:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2.2a}$$

$$y_t = f(W_{hy}h_t + b_y) \tag{2.2b}$$

donde x_t, h_t y y_t representan los vectores de activación de las capas de entrada, recurrente y de salida a tiempo t, respectivamente. El resto de los símbolos definen las matrices de pesos (W_{xh}, W_{hh} y W_{hy}), donde los subíndices representan que activación conectan; el vector de bías para cada capa (b_h y b_y) y la función de activación f. Nótese la recurrencia en la activación h_t. Su valor depende del estado previo h_{t-1} controlado por la matriz de pesos W_{hh}. Este tipo de modelos retiene perfectamente la información de una serie temporal, donde cada instante depende de los valores anteriores de dicha

serie. Sin embargo, desarrollar el descenso por gradiente con longitud, a priori, ilimitada requiere de alguna estrategia que haga posible el cálculo de los gradientes. Se muestra en la figura 2.6 un esquema ilustrativo de las conexiones de la RNN y el concepto de despliegue (*unfold*) habitual que resuelve este problema.

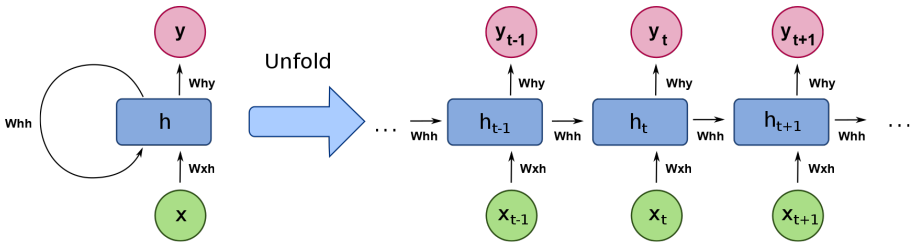


Figura 2.6: Ilustración del concepto de recurrencia en una RNN. [Imagen creada por *François Deloche* para Wikipedia - https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg]

En la parte izquierda de la figura, esta arquitectura de red tiene una única capa oculta h donde existe una conexión recurrente consigo misma, representado por la ecuación 2.2a. Por otro lado, se representa en la parte derecha de la figura el concepto de despliegue, que permite comprender esta arquitectura de red como una red neuronal multicapa con un número, a priori, infinito de capas intermedias. Es indispensable definir un límite de despliegue con el que establecer una longitud máxima de dependencia temporal.

Este documento se basa en dos modelos que utilizan como base esta arquitectura (ver sección 4.2). Sin embargo, existen otros modelos de redes neuronales recurrentes con arquitecturas más complejas, como las RNNs de segundo orden [41, 42], las *Long-Short Term Memory* (LSTM) [43] o las *Gated Recurrent Unit* (GRU) [11].

2.3.2. Redes neuronales con memoria aumentada

En los últimos años, continuando con la búsqueda del paralelismo entre las RNNs y los autómatas, se ha dado un paso más para aproximarlas al concepto teórico de una máquina de Turing: las redes neuronales con memoria aumentada (*Memory Augmented Neural Networks*) [19–21, 44, 45]. Estas redes típicamente utilizan una red neuronal recurrente estándar, como una RNN de Elman o una LSTM, a la que introducen algún tipo de mecanismo adicional, diferenciable, que actúa como memoria externa. Estos mecanismos permiten al modelo proporcionar mejores resultados en aquellos problemas que requieren un uso intensivo de la memoria. En estas situaciones, parece que la RNN se comporta como la CPU del sistema.

Uno de los mecanismos más interesantes es la Neural Turing Machine (NTM) [19, 45], que busca simular directamente el mecanismo de una máquina de Turing, tal y como se muestra en la figura 2.7. Como se puede observar, la NTM tiene un controlador, típicamente una LSTM, que, a partir de la entrada y la lectura en memoria, genera una escritura en memoria y la salida en cada instante de tiempo. La complejidad del modelo se basa en los métodos de lectura y escritura en memoria.

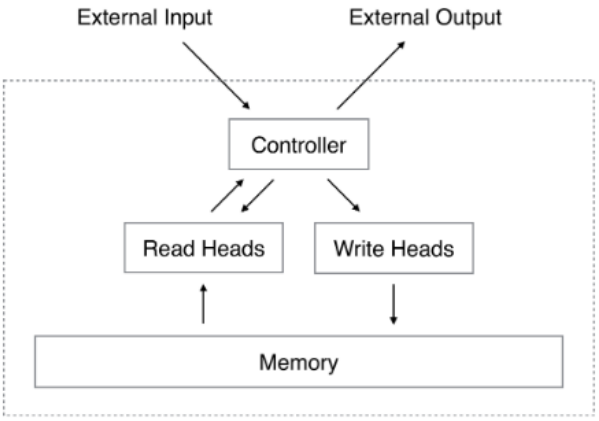


Figura 2.7: Arquitectura de una Neural Turing Machine. Imagen extraída del artículo original [19].

Este modelo tan prometedor ha dado excelentes resultados en tareas como copiar secuencias u ordenarlas. Sin embargo, aunque se sigue avanzando día a día, aún queda mucho por explorar con las arquitecturas MANN. En este proyecto se ha desarrollado una RNN basada en este concepto de memoria aumentada, donde la la capa recurrente de la red se encarga exclusivamente de procesar la información temporal de una secuencia (ver sección 4.2.2).

3

Estado del arte

Las técnicas de aprendizaje automático y, concretamente, las redes neuronales profundas se han vuelto esenciales para una gran cantidad de aplicaciones, tales como clasificación de imágenes [8, 9], generación de música [15], reconocimiento de voz [14], o procesamiento de lenguaje natural [10–12], entre muchas otras. Entrando en detalle, se ha demostrado que las redes neuronales recurrentes (RNNs) son un modelo Turing-completo [16], es decir, estas redes tienen la capacidad de procesar cualquier problema computable.

A lo largo de este capítulo se va a realizar un análisis del estado de la cuestión sobre este campo de investigación. De esta manera, este capítulo se divide en tres secciones: en la primera sección (3.1) se presenta un estudio sobre inferencia de gramáticas formales y aprendizaje de las mismas utilizando RNNs. En la segunda sección (3.2) se hace una breve revisión del trabajo realizado en los últimos años sobre el uso de las RNNs para problemas de predicción de series temporales, más concretamente sobre generación de lenguaje natural. Por último, en la sección 3.3 se discute la interpretabilidad de estos modelos.

3.1. Inferencia de gramáticas utilizando RNNs

Es posible demostrar que una RNN es un modelo Turing-completo [16], por lo que debe ser capaz de aprender cualquier lenguaje formal. Desde el planteamiento de esta idea en los años 90 [4], se ha estudiado en numerosas ocasiones la equivalencia entre estas redes y los modelos matemáticos abstractos que identifican estos lenguajes. Bien es cierto que en este campo de investigación se ha hecho hincapié en encontrar la equivalencia entre estas redes y los autómatas finitos a la hora de identificar lenguajes regulares [1–3, 5–7, 46]. Tal y como afirma H. T. Siegelmann [17], si una RNN es capaz de identificar un lenguaje regular, esta debería ser equivalente al autómata finito determinista (AFD) que lo identifica.

Siguiendo la revisión de H. Jacobsson realizada en 2005 [18], existe una abrumadora cantidad de artículos en los que se busca esta equivalencia, entre los que se encuentra no solo la extracción de reglas para transformar una RNN en un AFD, sino también otras técnicas como la *clusterización* [13,47] o la proyección del espacio de estados interno [1, 3, 6, 7, 48], normalmente utilizando algoritmos de reducción de dimensionalidad. Es interesante mencionar también una revisión reciente de Q. Wang [49], donde expone sus resultados en esta materia utilizando RNNs de segundo orden.

El enfoque principal de la extracción de reglas gramaticales para inferir lenguajes utilizando RNNs típicamente ha consistido en aplicar diversas técnicas de cuantización, que tratan de transformar el espacio interno de estados de la red en un conjunto discreto de estados que corresponden a los estados del AFD que se pretende extraer. Sin embargo, este hecho es un punto de conflicto en este campo de investigación, ya que, tal y como indica J. F. Kolen [50], el mero hecho de transformar un sistema continuo en discreto supone enfrentarse a un riesgo que puede provocar errores en la ejecución, ya que supone, a priori, una incoherencia. Tal y como comenta Jacobsson en su revisión [18] y si se realiza una comparativa con la inferencia clásica de gramáticas regulares [51], aunque los métodos de inferencia de las RNNs son capaces de generar un AFD equivalente al lenguaje que se pretende inferir, se ha demostrado empíricamente que estas redes presentan un problema de generalización cuando se aplican a cadenas que son significativamente diferentes al conjunto de entrenamiento de la red, lo cual refuerza la argumentación de Kolen [50].

Sin embargo, en este ámbito, ya que estas redes son capaces de aprender los casos habituales y solo presentan problemas con cadenas poco frecuentes o muy diferentes a las vistas durante el entrenamiento, son una herramienta aceptada por la comunidad científica como método de extracción de reglas del lenguaje. Es más, en la mayoría de los casos las reglas extraídas generalizan mejor que las propias RNNs [49].

3.2. Modelado de lenguaje natural

El procesamiento de lenguaje natural (NLP) es un campo de la ciencia de la computación, la inteligencia artificial y la lingüística que estudia las interacciones entre la informática y el lenguaje humano. En esta área de investigación, hay muchos enfoques por los cuales el NLP puede ser abordado: análisis y clasificación de diálogos, que es la tarea de clasificar un enunciado respecto a la función que cumple en el diálogo [52], corrección gramatical de texto [53], análisis de sentimiento [54] o generación de texto [55, 56], entre otras.

El área principal que se abarca en este proyecto consiste en la predicción de series temporales. Haciendo énfasis en la generación de texto, que se basa en la predicción del siguiente elemento de la secuencia, este problema se divide en dos aproximaciones: (1) generación a nivel de palabra, donde la red recibe como entrada los identificadores de cada palabra y debe predecir la siguiente, o (2)

generación a nivel de carácter, donde la red recibe la entrada símbolo a símbolo y debe predecir el siguiente. La función de coste utilizada en este tipo de problemas consiste en reducir un valor de perplejidad (*perplexity*), utilizado en la generación a nivel de palabra, y el BPC (*bit per character*) cuando se genera a nivel de carácter. Ambas métricas son pequeñas variaciones de la clásica *cross-entropy*, tal y como se muestra en las siguientes ecuaciones:

$$perplexity = 2^{-crossEntropy} \tag{3.1a}$$

$$bpc = -\frac{1}{N}\sum_{i}^{N} log_2 \hat{P}(x_i) \tag{3.1b}$$

$$crossEntropy = -\frac{1}{N}\sum_{i}^{N} log \hat{P}(x_i) \tag{3.1c}$$

De esta manera, el estado del arte más actual se basa, mayoritariamente, en aplicar ciertas modificaciones sobre la clásica LSTM [43] para alcanzar resultados cada vez más extraordinarios. Arquitecturas altamente complejas como la Mogrifier LSTM [55], la AWD-LSTM (ASGD Weight-Dropped LSTM) [57] y sus variaciones AWD-LSTM-DOC [58] y AWD-LSTM-MoS [59], acompañadas de otras estrategias complejas de regularización, han proporcionado resultados sorprendentes respecto a la capacidad de aprendizaje de generación de texto. Todas las estrategias coinciden con el uso de la evaluación dinámica [60], que consiste, básicamente, en que el modelo debe continuar aprendiendo cuando se está evaluando, ya que, al procesar una secuencia λ_{t+1}, el modelo habrá observado la secuencia completa hasta λ_t. Esta estrategia mejora considerablemente el resultado en validación y test, por lo que se ha convertido en un estándar por parte de la comunidad científica.

3.3. Interpretabilidad

Las técnicas de aprendizaje profundo se han vuelto esenciales para una gran cantidad de aplicaciones. Sin embargo, incluso hoy en día, estos modelos profundos no han terminado de encontrar su lugar en muchas aplicaciones comerciales debido a que aproximaciones más sencillas, como modelos lineales o árboles de decisión, proporcionan una mayor interpretabilidad [61]. Es por este motivo que las técnicas de interpretación de estos modelos de aprendizaje profundo se han vuelto bastante populares entre la comunidad científica.

En áreas como la clasificación de imágenes se utilizan técnicas de análisis de sensibilidad, estudiando el gradiente de forma local para detectar máximas variaciones, o técnicas conocidas como *Backward propagation* (Layer-wise Relevance Propagation [61]), donde, partiendo de la capa de salida, se transmite hacia atrás la relevancia que tiene la información de cada capa para determinar la respuesta del modelo. Estos algoritmos, en definitiva, indican qué píxeles de la imagen son los que

determinan con mayor relevancia la decisión de la red. Por otro lado, respecto al procesamiento de secuencias, como música [62] o lenguaje natural [63] se han desarrollado nuevas técnicas de interpretabilidad, donde se analiza la activación de la capa oculta frente al tiempo buscando neuronas que identifiquen patrones con algún significado relevante.

Respecto al tema que concierne a este proyecto, no existe actualmente una interpretación que pueda demostrar que las RNNs estén implementando directamente un AFD. A. Karpathy [63] demuestra, utilizando una LSTM, que estas redes son especialmente eficientes cuando tratan de generar lenguaje natural y, además, analiza del comportamiento interno de la red demostrando que existen ciertas neuronas fácilmente interpretables que tienen la capacidad de reconocer ciertos patrones en el problema. Además, se ha demostrado que introducir ciertas modificaciones durante el entrenamiento de las RNNs [6,7] permite explorar su interpretabilidad cuando se enfrentan a aprender problemas regulares. La inferencia de gramáticas es, en definitiva, otra aproximación para interpretar el comportamiento de una RNN, ya que, si es posible afirmar que una red es equivalente a un autómata, es posible hacer una sustitución entre ambos modelos sin que se vea afectado el resultado final [16] y, por tanto, directamente se podría analizar el comportamiento de la red.

4

Diseño

El objetivo de este proyecto consiste en explorar la interpretabilidad de las redes neuronales recurrentes buscando una equivalencia directa con los autómatas finitos deterministas. Si es posible encontrar una equivalencia clara entre ambos, la interpretación también queda resuelta. Después de haber definido los conceptos básicos que el lector debe conocer (capítulo 2) y haber estudiado el estado de la cuestión sobre el procesamiento de lenguaje natural y, de una forma más genérica, las técnicas de visualización e interpretabilidad (capítulo 3), corresponde desarrollar la descripción de los conjuntos de datos a utilizar (sección 4.1) y el diseño de las arquitecturas de RNNs implementadas (sección 4.2).

4.1. Descripción de los datos

A lo largo del desarrollo de este proyecto ha surgido la necesidad de utilizar diferentes conjuntos de datos para los experimentos que se han planteado. De esta forma, se ha generado un conjunto de lenguajes regulares con los que comenzar la investigación, descrito en la sección 4.1.1. Además, como alternativa a un problema de clasificación, se ha generado un dataset para predecir expresiones algebraicas bien formadas (sección 4.1.2). Por último, para aplicar los modelos estudiados a dos problemas reales basados en el procesamiento de lenguaje natural, se han utilizado el texto completo del Quijote, de Miguel de Cervantes, en español, la obra literaria más influyente del Siglo de Oro (sección 4.1.3) y un conjunto de opiniones de usuarios sobre películas de la plataforma IMDB [64] (sección 4.1.4).

4.1.1. Lenguajes regulares

Lenguajes regulares sobre el alfabeto {a, b}

En este proyecto se han considerado los lenguajes regulares sobre el alfabeto {a, b} definidos en la tabla 4.1, que incluye los siguientes problemas: dos problemas iniciales con los que se inició el trabajo (*Paridad* y *BxA*) y las bien conocidas *Tomita Grammars* [65], que son consideradas un estándar de referencia respecto a la inferencia de lenguajes regulares.

Nombre	Lenguaje regular	Expresión regular
Paridad	Cadenas con paridad de a's.	b*(ab*ab*)*
BxA	Cadenas que empiezan por b y terminan por a.	b(a+b)*a
Tomita1	Cadenas con solo a's.	a*
Tomita2	Cadenas con solo secuencias de ab.	(ab)*
Tomita3	Cadenas sin un número impar de a's seguidas por un número impar de b's.	b*[aa(aa)*b*+a(aa)*bb(bb)*]* (a+λ)
Tomita4	Cadenas con un número menor que 3 b's consecutivas.	(a+ba+bba)*(bb+b+λ)
Tomita5	Cadenas de longitud par con un número par de a's.	[aa+bb+(ab+ba)(aa+bb)* (ab+ba)]*
Tomita6	Cadenas con diferencia de a's y b's múltiplo de tres.	[ba+(a+bb)(ab)*(b+aa)]*
Tomita7	Cadenas con la forma $X_1Y_1X_2Y_2$ donde $X_{1,2}$ son dos subcadenas con solo b's e $Y_{1,2}$ son dos subcadenas con solo a's.	b*a*b*a*

Tabla 4.1: Descripción de los lenguajes regulares utilizados en este trabajo. A las clásicas *Tomita Grammars* [65] se añaden los problemas de *Paridad* y *BxA*.

Para tratar las cadenas del lenguaje como una serie temporal, las dos arquitecturas de red desarrolladas durante este trabajo, descritas en la sección 4.2, utilizan una arquitectura Many-to-Many, donde un elemento de la serie en un instante determinado X_t da la respuesta correspondiente en ese instante Y_t. Por este motivo, se ha decidido introducir un símbolo de escape $\$$ para identificar la separación de cada cadena, por lo que puede interpretarse como el inicio (o final) de la cadena. La introducción de este símbolo permite que la red sea capaz de procesar el conjunto de cadenas como una serie temporal, sin interrupción y sin necesidad de establecer un tamaño máximo ni de reiniciar el estado de la red con cada nueva cadena.

Para la generación de los conjuntos de entrenamiento y test se ha desarrollado un generador de cadenas aleatorias con los símbolos del alfabeto, incluyendo el símbolo de escape, y las probabilidades de cada uno de los símbolos. La tabla 4.2 describe cada una de las configuraciones de los conjuntos de entrenamiento y validación del modelo. Para ello, se han generado seis datasets que serán empleados durante algunos experimentos: un conjunto de entrenamiento (*train*) que contiene 50.000 símbolos con la misma probabilidad de aparición de los símbolos a y b y varios conjuntos de test con diferentes características: el conjunto *big* se ha generado con las mismas condiciones que el conjunto de entrenamiento, excepto su longitud (100.000 símbolos). El conjunto *long* contiene únicamente 20.000 símbolos y tiene una menor probabilidad de aparición del símbolo $\$$, lo que implica que las cadenas sean más largas que en el conjunto de entrenamiento. Los conjuntos *all as* y *all bs*

representan el mismo concepto pero con símbolos contrarios: ambos tienen 15.000 símbolos y una elevada probabilidad de aparición del símbolo a o b, respectivamente. Por último, el conjunto *mega*, que contiene 100 cadenas diferentes de longitud 1.000.000, tiene una probabilidad de aparición del símbolo $ de 0. Este salto cuantitativo se utilizará para demostrar la capacidad de generalización de las redes y su estabilidad.

Dataset	chars	a prob.	b prob.	$ prob.	mean len	min len	max len
train	50.000	0,45	0,45	0,1	8,9	0	95
big	100.000	0,45	0,45	0,1	9,0	0	81
long	20.000	0,495	0,495	0,01	88,3	0	477
all as	15.000	0,98	0,01	0,01	113,5	0	566
all bs	15.000	0,01	0,98	0,01	95,8	0	475
mega	1.000.000	0,5	0,5	0	1M	1M	1M

Tabla 4.2: Descripción de los datasets utilizados para los lenguajes regulares sencillos descritos. Se muestran las probabilidades de cada símbolo para cada configuración y el máximo, mínimo y media de la longitud de las cadenas generadas.

Todos estos datasets se han generado de la siguiente manera: (1) un fichero de entrada con una serie aleatoria con los símbolos a, b y $ con las probabilidades de la tabla 4.2 y (2) un fichero de salida con la correspondiente respuesta del autómata finito determinista para cada instante de tiempo con cada símbolo.

Suma en diferentes bases

El siguiente conjunto de lenguajes regulares consiste en el problema de la suma de dos sumandos en diferentes bases (2, 4 y 10). En este caso, los lenguajes se han generado, por ejemplo, sobre el alfabeto {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} en el caso de la suma en base 10. El símbolo de escape $ vuelve a jugar papel crucial al identificar la separación entre sumas. La generación de los conjuntos de entrenamiento y test ha consistido en generar, de forma completamente aleatoria, dos conjuntos de entrada X_1 y X_2 de longitud 200.000, con el que se identifican los dos sumandos; y un conjunto de salida Y, del mismo tamaño, donde se imprime el resultado de la suma en ese instante de tiempo, teniendo en cuenta el acarreo. De esta forma la red recibe dos dígitos como entrada para dar la suma como resultado. Se muestra en la tabla 4.3 un ejemplo de los ficheros de entrada y salida con base 2. Cabe destacar que los dígitos han sido invertidos para que la cifra menos significativa sea el primer dígito visto, de forma contraria a la representación tradicional.

Input 1	$01011101010111010$01011101
Input 2	$11010101101010110$01011010
Output	01010011000001100100101000

Tabla 4.3: Ejemplo de los primeros 26 símbolos de los dos ficheros de entrada y el fichero de salida para la suma en base 2. Los símbolos $ aparecen en los dos ficheros de entrada en los mismos instantes de tiempo y la salida esperada en esos instantes es el acarreo del último dígito.

Este conjunto de problemas tiene la dificultad añadida de que cuanto mayor sea la base en la que entrena a la red, mayor es la complejidad del modelo, ya que el número de neuronas de entrada se define como la concatenación de dos vectores en formato One-hot representando cada uno de los dígitos de entrada. También sucede que la complejidad del autómata finito que identifica el problema aumenta cuanto mayor sea la base. El resultado del análisis de esta afirmación es crucial para entender la principal diferencia entre las dos arquitecturas de red que se van a estudiar (ver sección 5.2.2).

4.1.2. Expresiones algebraicas

El siguiente problema cambia el planteamiento y se aproxima a los problemas reales que se tratan en los siguientes experimentos. Este consiste en generar expresiones algebraicas. Las redes son entrenadas para predecir el siguiente símbolo en una expresión algebraica, incluyendo los operadores ($*$, $/$, $+$ y $-$), paréntesis de apertura y cierre y los operandos, representados por el símbolo a. Se incluye, como en los problemas anteriores, el símbolo $ como separador, lo que implica que la cadena generada debe ser correcta. Uno podría argumentar con razón que el lenguaje que identifica las expresiones algebraicas no es un lenguaje regular. Sin embargo, si se fija la profundidad máxima de paréntesis, es decir, se limita en el conjunto de datos el nivel de apertura de paréntesis a un valor π, el lenguaje es regular.

Como en el caso anterior, se ha desarrollado un generador de expresiones algebraicas aleatorias que cumplen las reglas gramaticales descritas en la tabla **2.2**. Siguiendo esta estrategia, los subconjuntos de entrenamiento y test se definen con una única cadena con 200.000 símbolos generados aleatoriamente. Se muestra en la tabla 4.4 los primeros 156 símbolos del conjunto de entrenamiento. Como se puede observar, cualquier cadena entre dos símbolos $ es una expresión aritmética válida con profundidad menor o igual que 5.

$((a)/a/a)+a+(a-a-a+((((a)))/(((a))+a/a)))$(a/(a-a-((
(a-a*a+a))/a)-((a/(a)/(a/a/a)))+a))$a-(((a/a*a)+a)-a)
-(a*(a)/(a))a(((((a)-(a-a+a-a)/a/a+a)/((a*a/a)/(...

Tabla 4.4: Primeros 156 símbolos del conjunto de entrenamiento del problema de generación de expresiones algebraicas.

En este problema se evalua el modelo entrenado midiendo la capacidad de generar expresiones aritméticas correctas: (1) empezando por un símbolo $, la red genera 50.000 símbolos, (2) la cadena completa se divide por el símbolo $ en las subcadenas correspondientes, (3) se comprueba, utilizando las reglas gramaticales anteriores, si cada cadena es correcta y (4) se define el acierto del modelo como el porcentaje de cadenas bien generadas. Se muestran en la tabla 4.5 algunos ejemplos, donde solamente la última cadena es correcta. La primera está mal formada ya que hay dos operadores contiguos. La segunda cadena falla ya que hay un paréntesis vacío y las dos siguientes cadenas fallan ya que la profundidad de apertura y cierre de los paréntesis es errónea.

Ejemplo	Test	¿Por qué?
$a-((a)+a)*/a+(a)$	$\times$	$*/$
$(a+())-a*(((a)))$	$\times$	$()$
$a-((a))+a)/a+(a)$	$\times$	Depth = -1
$((a(a))+a*a/((a))$	$\times$	Depth = +1
$((a(a))+a*a/((a)))$	$\surd$	

Tabla 4.5: Ejemplos de cadenas correctas o incorrectas generadas por una red entrenada para generar expresiones algebraicas.

4.1.3. El Quijote

¿Quién no conoce las aventuras de la novela literaria más prestigiosa del Siglo de Oro español e, incluso, de toda la literatura española? *El Ingenioso Hidalgo Don Quijote de La Mancha*, de Miguel de Cervantes, será el punto de partida para poner a prueba las arquitecturas desarrolladas en el procesamiento de lenguaje natural (NLP) y, más concretamente, la generación de texto en español. En este problema, los caracteres se introducen en la red sin ningún tipo de preprocesado, exceptuando la división en los conjuntos de entrenamiento (70 %), validación (15 %) y test (15 %). El modelo se entrena para generar texto a nivel de caracter, es decir, intentar predecir el símbolo siguiente. La tabla 4.6 resume los conjuntos de datos utilizados, describiendo algunas de sus características, como el número de caracteres, el número total de palabras o el número de palabras diferentes.

Dataset	chars	palabras	palabras únicas
Entrenamiento	726.877	131.132	18.054
Validación	155.760	28.175	6.185
Test	155.760	27.712	6.162

Tabla 4.6: Descripción de los datasets utilizados para la generación de texto en español. Incluye la longitud, el número total de palabras y el número de palabras diferentes en cada subconjunto.

4.1.4. Opiniones de películas de IMDB

Este último conjunto de datos surge de una última aplicación de la red Dual, la segunda arquitectura de red desarrollada en este proyecto (ver sección 4.2.2), donde se ha decicido utilizar el modelo para aplicar Opinion Mining o, lo que es lo mismo, Sentiment Analysis, es decir, extraer a partir de un texto u opinión, el valor de esa opinión (si es negativo o positivo). En este problema en concreto, la opinión no es más que la puntuación que el usuario da a la película. Los datos de este conjunto consisten en las páginas HTML en crudo de las opiniones de las películas visitadas por un crawler implementado en la asignatura de *Minería Web* para extraer directamente las opiniones de la plataforma, por lo que se requiere un preprocesamiento de los datos antes de realizar el análisis de sentimiento. En la tabla 4.7 se muestra un resumen descriptivo de los conjuntos de datos utilizados, donde se incluye el número de opiniones, la longitud media de las opiniones tanto antes como después del preprocesamiento, el número de términos tanto antes como después del preprocesamiento y la media de las puntuaciones.

	Entrenamiento	**Test**
# Opiniones	671	168
Longitud media antes	1546,26	1372,76
Longitud media después	129,91	113,33
# Términos antes	7021	5050
# Términos después	5242	4136
Puntuación media	8,48	8,10

Tabla 4.7: Descripción de los conjuntos de datos utilizados para aplicar análisis de sentimiento. Se muestra el número de opiniones en cada dataset, la longitud media de las opiniones tanto antes como después del preprocesamiento, el número de términos antes y después del procesamiento y la puntuación media.

4.2. Modelos

Una vez se han definido los problemas que se han planteado a lo largo de este proyecto, junto a los conjuntos de datos que se han utilizado, se van a describir los modelos que se han desarrollado. En la sección 4.2.1 se describe una modificación de la red de Elman clásica [13] con ciertas modificaciones y, por otro lado, en la sección 4.2.2, se desarrolla una nueva arquitectura, la red Dual, que surge de la idea de combinar el modelo anterior con una capa feedforward adicional para simplificar la complejidad de la capa recurrente, que actúa como un tipo de memoria externa de la red. Ambos modelos han sido implementados con la librería *Keras* [22], una potente librería de $Python$ basada en *Tensorflow* enfocada al diseño de modelos de aprendizaje profundo, con el fin de poder extender el trabajo si fuese necesario. El código de la implementación se puede encontrar en el apéndice A.

4.2.1. Red Neuronal Recurrente de Elman

La red de Elman [13] es la arquitectura de red neuronal recurrente más sencilla. Como se ha descrito en la sección 2.3.1, esta añade una relación de dependencia temporal ya que introduce una conexión recurrente en la capa intermedia. En trabajos previos [6, 7] se utiliza una variante de esta red donde se introduce ruido en la función de activación, controlado por el estado interno del instante anterior, regido por las ecuaciones siguientes:

$$h_t = tanh(W_{xh}x_t + W_{hh}h_{t-1} + X_\nu \circ h_{t-1} + b_h) \tag{4.1a}$$

$$y_t = \sigma(W_{hy}h_t + b_y) \tag{4.1b}$$

donde X_ν es una variable Gaussiana centrada en 0 y con varianza ν. De esta manera se introduce un ruido Gaussiano de forma proporcional al estado de la neurona en el instante de tiempo anterior h_{t-1}. El operador $\circ$ representa la multiplicación elemento a elemento. Este método permite al modelo introducir al mismo tiempo ruido y regularización sin que estos entren en conflicto. Durante el aprendizaje, tanto el ruido como la regularización se pueden introducir de forma constante (el método tradicional) o de forma adaptativa o incremental, es decir, a medida que aumentan las épocas durante el entrenamiento, tanto el factor de ruido ν como el factor de regularización L1 aumentan, siguiendo la siguiente ecuación:

$$ruido = min(\nu, \frac{epoca \cdot pendiente \cdot \nu}{max\_epocas}) \tag{4.2}$$

donde max_epocas es el número de épocas de entrenamiento. Es necesario definir una $pendiente$ para indicar la velocidad de crecimiento del ruido hasta llegar al máximo. Esta ecuación aplica de forma equivalente a la regularización L1 adaptativa. De esta forma, en las primeras épocas tanto el ruido como la regularización tienen poco peso en el coste final, perimitiendo al modelo aprender con mayor facilidad.

4.2.2. Red Dual

Como se muestra en la sección 5.2.2, se ha comprobado que los modelos de redes recurrentes tienen la particularidad de que necesitan mantener la información temporal que extraen de la secuencia de entrada y, además, deben procesar esa información para dar la respuesta final. Este comportamiento implica que una única capa interna debe tener la capacidad tanto de recordar como de proporcionar la respuesta del modelo, lo cual aumenta considerablemente su complejidad global y, por supuesto, su posible interpretabilidad.

La segunda arquitectura de red que se ha desarrollado en este proyecto, la cual se ha denominado red Dual, surge de la idea de separar la información puramente temporal de la secuencia de entrada del resto de la información, utilizando una capa recurrente e introduciendo una capa feedforward que recibe como entrada la concatenación de la salida de esta capa recurrente con la entrada de la red en el instante actual, tal y como se describe con las siguientes ecuaciones:

$$h_t = tanh(W_{xh}x_t + W_{hh}h_{t-1} + X_\nu \circ h_{t-1} + b_h) \tag{4.3a}$$

$$c_t = tanh(W_{xc}x_t + W_{hc}h_t + b_c) \tag{4.3b}$$

$$y_t = \sigma(W_{cy}c_t + b_y) \tag{4.3c}$$

Esta implementación, mostrada gráficamente en el diagrama 4.1, se basa en la idea descrita anteriormente, donde la capa recurrente aprende a solamente procesar la información que debe mantener a lo largo del tiempo, actuando como una memoria interna, mientras que la capa feedforward recibe información de esa memoria y de la entrada en el instante actual para proporcionar la respuesta final del modelo. Esta división de procesamiento motiva el nombre *Dual* del modelo. Las modificaciones descritas para el modelo anterior, donde se introduce un factor de ruido ν y regularización L1 adaptativos también aplican en esta arquitectura Dual, ya que mantienen el mismo efecto.

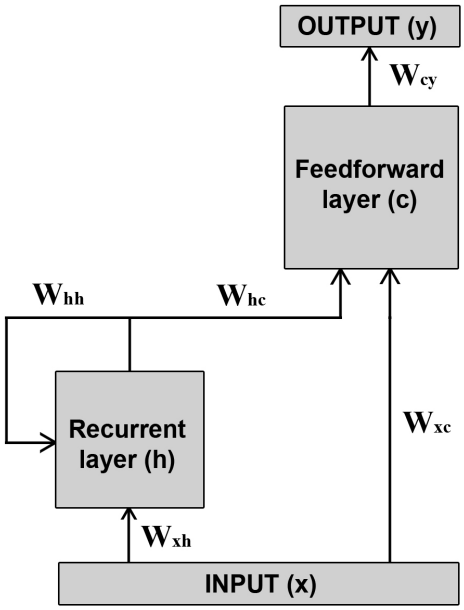


Figura 4.1: Diagrama de la arquitectura de la red Dual.

5

Desarrollo, análisis y resultados

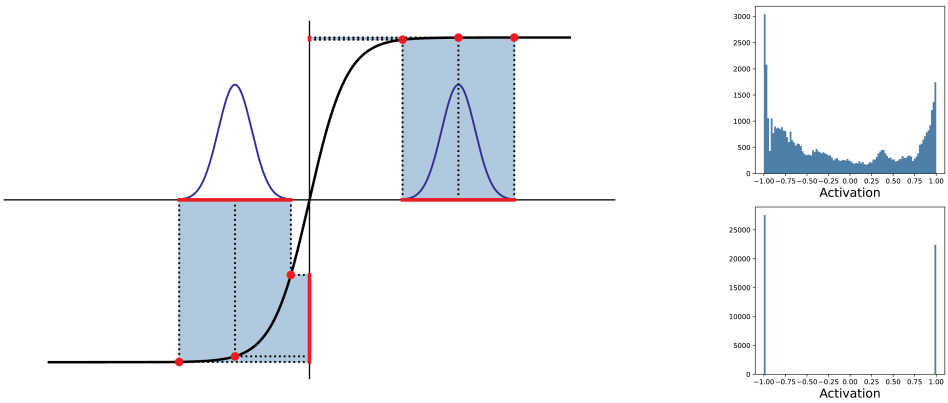
Una vez se han descrito los objetivos, los conceptos básicos, el estado del arte y el diseño de los modelos, el objetivo de este capítulo consiste en presentar el desarrollo de los experimentos y el análisis de los resultados obtenidos a lo largo del proyecto. En general, se han realizado pruebas sobre los modelos descritos en las secciones 4.2.1 y 4.2.2 y, en algunas ocasiones, se incorpora una LSTM estándar con el fin de tener una referencia. Este capítulo se divide en las siguientes secciones: en la sección 5.1 se lleva a cabo un análisis empírico de la estabilidad y la capacidad de generalización de la Noisy Elman RNN. En la sección 5.2 se realiza un estudio completo de la nueva arquitectura de red Dual, comparando los resultados con la Noisy Elman RNN y la LSTM. En la sección 5.3 se introduce la entropía como métrica de interpretabilidad de un modelo. Por último, en la sección 5.4 se aplica el modelo Dual a dos problemas con mucha relevancia en el estado del arte actual sobre el procesamiento de lenguaje natural: la generación de texto y el análisis de sentimiento.

5.1. Análisis de estabilidad

En este primer estudio se pretende demostrar la estabilidad de la red de Elman modificada al ser interpretada como un autómata finito. Encontrar un modelo estable y robusto lleva intrínseco el concepto de generalización, donde pequeñas variaciones en los datos de entrada no provocan, incluso a lo largo del tiempo, cambios considerables en la respuesta final del modelo. De esta forma, teniendo en cuenta esta consideración, el primer experimento consiste en realizar un estudio empírico de la estabilidad de la red cuando se introduce una cierta cantidad de ruido en la función de activación, pretendiendo comprender tanto su comportamiento como el resultado final que se alcanza. Siguiendo estos objetivos, esta primera sección se divide en tres apartados: en la sección 5.1.1 se realiza un análisis del efecto del ruido en la función de activación de la capa recurrente, justificando el comportamiento del modelo. En la sección 5.1.2 se muestra una interpretación visual del espacio de estados interno del modelo, mostrando la generación interna de clusters estables como si de estados finitos se tratase. Por último, en la sección 5.1.3 se realiza un análisis de la estabilidad del modelo y de su capacidad de generalización.

5.1.1. Efecto del ruido

En el estado del arte del aprendizaje automático se ha demostrado que el ruido puede utilizarse tanto como un factor de regularización [66, 67] como un método de escape de la zona de saturación [68], lo que permite un mejor aprendizaje incluso utilizando funciones de activación cuya derivada es 0 en algún rango, como son las funciones *hard-sigmoid*, *hard-tanh* o *relu*. En esta sección se muestra que la inyección de ruido puede incluso tener el efecto contrario y forzar a las neuronas a trabajar en el régimen de saturación [6, 7]. Se muestra en la figura 5.1 el efecto que tiene introducir un ruido Gaussiano en la pre-activación de una función *tanh*. Si una neurona con esta función de activación necesita dar una respuesta estable, necesita desplazar su actividad a la zona de saturación, o dicho de otro modo, moverse donde la pendiente tiene un valor cercano a 0.



(a) Efecto del ruido

(b) Activación de neuronas

Figura 5.1: Efecto del ruido en un modelo con activación *tanh*. En la figura de la izquierda (5.1(a)) se muestra el efecto de introducir ruido en la pre-activación de una *tanh*. Se pretende ilustrar cómo, mientras que en la región de saturación introducir ruido no perturba la salida, cuando la neurona trabaja cerca de la región lineal, una pequeña cantidad de ruido en la entrada provoca grandes diferencias en la salida. En la figura de la derecha (5.1(b)) se muestra un histograma de la activación de una neurona, tanto cuando el ruido no se utiliza (arriba) como cuando se introduce ruido $\nu = 1{,}0$ (abajo). Nótese la diferencia en la escala de los histogramas.

Sin embargo, es necesario tener otro factor en cuenta: si la red se entrena con regularización L1, esta penalización y el ruido tienen efectos contrarios. Mientras que la regularización pretende reducir los pesos a 0, cuya consecuencia directa es anular la activación, es decir, llevar a la neurona a su activación de máxima pendiente, el ruido sigue el efecto contrario, es decir, pretende llevar a la neurona a las zonas de saturación. Es por este motivo que la inyección de ruido en los modelos está controlada por la activación de la neurona en el estado anterior h_{t-1}, ya que cuando una neurona está regularizada ($h_{t-1} \approx 0$), el ruido es absolutamente contraproducente.

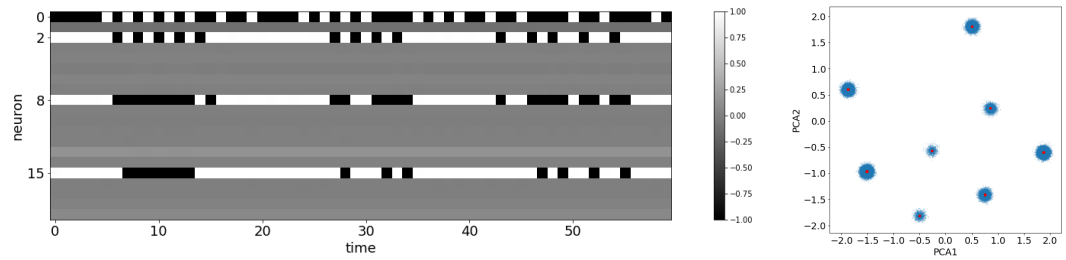
En la figura 5.1(b), por otro lado, se muestra un histograma típico de la activación de una neurona cuando un modelo se entrena sin ruido (histograma superior) o con ruido (histograma inferior) con el fin de ilustrar este comportamiento. Como se puede observar, cuando no hay ruido, la activación a lo largo del tiempo se distribuye en todo su rango de acción [-1, 1], mientras que cuando se introduce una cantidad suficiente de ruido, la activación de la neurona está claramente saturada en los extremos. Este comportamiento provoca que, en la mayoría de los casos, sea posible interpretar y analizar cada una de las neuronas por separado o como un conjunto de neuronas binarias.

5.1.2. Espacio de estados binario

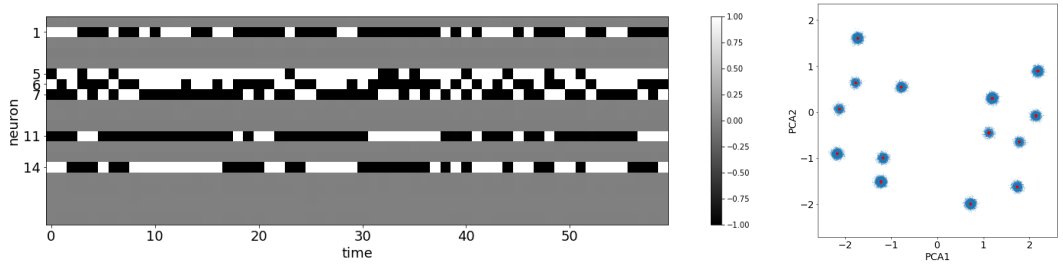
Los experimentos que se han realizado demuestran que entrenando una Noisy Elman RNN con los parámetros de ruido y regularización apropiados, el modelo binariza completamente los estados internos independientemente del número de neuronas de la capa oculta y, además, la regularización anula todas aquellas neuronas que son redundantes o innecesarias [6,7]. La red es entrenada para identificar los problemas regulares sobre el alfabeto $\{a, b\}$ descritos en la sección 4.1.1 y la suma en diferentes bases para llevar a cabo un análisis empírico de la estabilidad y su interpretación. Los resultados que se muestran a continuación obtienen un 100 % de acierto tanto en los conjuntos de entrenamiento como los diferentes conjuntos de validación para todos los problemas. Las redes entrenadas tienen 20 unidades en la capa oculta y son entrenadas durante 500 épocas con un factor de aprendizaje $l = 2{,}5$ y clip $c = 0{,}002$ [1] para minimizar la función de coste cross-entropy (ec. 3.1c) utilizando el descenso por gradiente estándar. El factor de ruido constante es $\nu = 0{,}6$ y el factor de regularización $r = 0{,}0004$, también constante, se aplica exclusivamente a los pesos, no a los bias. Concretamente, se muestra en la figura 5.2 un ejemplo de la activación de la capa oculta de dos modelos entrenados con los problemas *Tomita 3* y la suma en base 4, aunque es necesario mencionar que se ha observado un comportamiento similar en el resto de problemas.

Como se puede observar, solamente un conjunto de las neuronas está activo (las neuronas $\{0, 2, 8, 15\}$ en el caso del problema *Tomita 3* y las neuronas $\{1, 5, 6, 7, 11, 14\}$ en el caso del problema de la suma en base 4) de forma completamente binarizada, mientras que el resto mantienen una activación constante próxima a 0. En las gráficas de la derecha se utiliza el algoritmo PCA para extraer las dos primeras componentes principales y proyectar el espacio interno de la red a dos dimensiones. En esta proyección, se puede observar cómo el espacio de estados interno completo forma ciertos clusters (8 y 14 grupos, respectivamente) que corresponden a los diferentes vectores de activación únicos que se forman en un instante de tiempo determinado, es decir, cada una de las columnas de la figura izquierda.

[1] Utilizar un factor de clip consiste en limitar el gradiente para evitar que explote cuando se aplica un descenso por gradiente estándar.



(a) Tomita 3

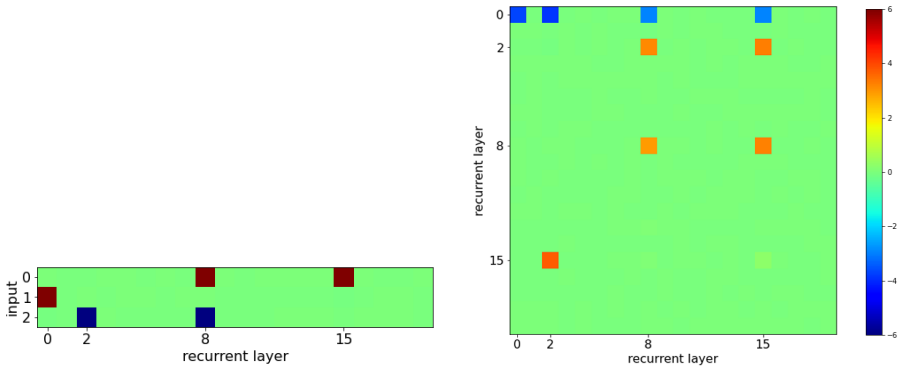


(b) Suma en base 4

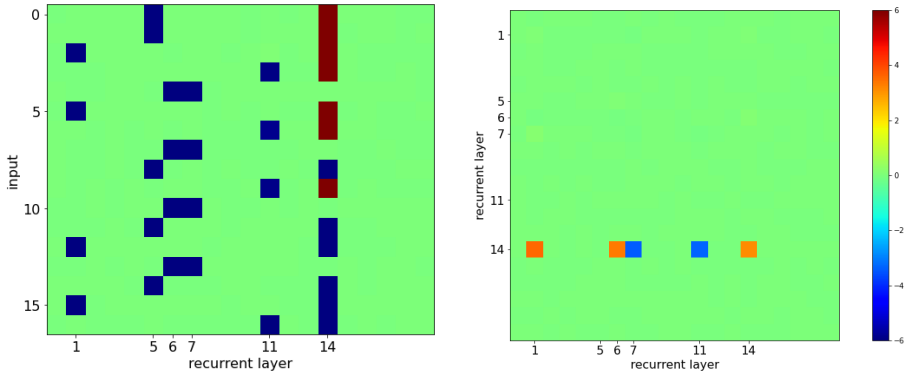
Figura 5.2: A la izquierda de cada figura se muestra la activación de las 20 neuronas de la capa oculta a lo largo de 60 unidades de tiempo (símbolos). A la derecha se muestra la proyección a 2 dimensiones de las dos componentes principales utilizando el algoritmo de PCA sobre 50.000 y 100.000 símbolos para los problemas mencionados, respectivamente. Es necesario destacar que los puntos de la proyección PCA han sido expandidos con cierto ruido Gaussiano para que se visualice con mayor claridad. La figura 5.2(a) muestra la activación de las neuronas y su proyección para el problema *Tomita 3*, mientras que la figura 5.2(b) lo hace para el problema de la suma en base 4.

Es importante destacar que, con el fin de visualizar fácilmente los clusters, se ha sumado un ruido Gaussiano para aumentar el tamaño de los clusters que se observan en la figura. Sin embargo, la transformación a 2 dimensiones genera exclusivamente los centros de cada cluster, lo que nos indica la discretización del conjunto de validación completo, que en el caso del problema *Tomita 3* contiene 50.000 estados y en el caso de la suma en base 4, 100.000.

Por otro lado, el análisis y la visualización de los pesos es otro método para comprender el comportamiento de una red neuronal. Solamente los pesos que interactúan con aquellas neuronas activas (no regularizadas) son distintos de 0, tal y como se expone en la figura 5.3, donde se muestran en un diagrama en escala de color las matrices de pesos de las capas de entrada a la capa recurrente W_{xh} y la capa recurrente W_{hh}.



(a) Tomita 3



(b) Suma en base 4

Figura 5.3: A la izquierda de cada figura se muestra la matriz de pesos de la capa de entrada a la capa recurrente W_{xh}. A la derecha se muestra la matriz de pesos de la capa recurrente W_{hh}. La figura 5.3(a) muestra los pesos para el problema *Tomita 3*, mientras que la figura 5.3(b) lo hace para el problema de la suma en base 4.

Como se puede observar, solamente las neuronas activas tienen pesos significativamente mayores que 0. Además, una observación interesante en el problema de la suma (figura 5.3(b)), que más adelante ha conducido al desarrollo de la red Dual, es que solamente una neurona de la capa recurrente (neurona 14) proyecta la información que contiene sobre esta misma capa. El resto de neuronas activas recibe la información exclusivamente de la entrada y esta neurona (14) y la propaga hacia adelante como si de un modelo feedforward se tratase.

Por último, ya que un conjunto de clusters representa el espacio de estados interno del modelo, analizar las transiciones entre cada uno de ellos según cada uno de los símbolos de entrada es, posiblemente, el siguiente paso que uno podría llegar a analizar. La respuesta a esta cuestión es que, efectivamente, las transiciones entre clusters son deterministas. Si se inicializa el estado del modelo a un cluster concreto y se observa la salida generada para cada símbolo de entrada, o, lo que es lo mismo, se analiza el estado interno en el instante siguiente h_{t+1} siempre que el símbolo de entrada x_{t+1} sea el mismo, se puede ver que todos los estados de salida pertenecen a un mismo cluster. De hecho, la red transita entre sus estados de la misma forma en que lo haría un autómata finito cuando se procesa una cadena de entrada. Se muestra en la figura 5.4 un ejemplo de transiciones desde un mismo estado para el problema *Tomita 3*. En el apéndice B se muestran todas las transiciones de la red desde cada uno de los estados de este ejemplo.

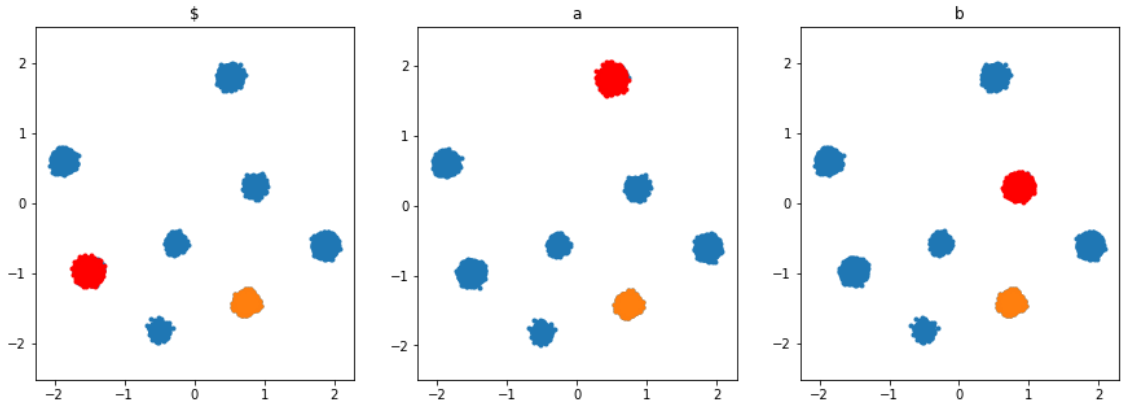


Figura 5.4: Ejemplo de las transiciones de los estados h_t pertenecientes al cluster coloreado en naranja con cada símbolo $\{\$, a, b\}$ para el problema *Tomita 3*. El cluster rojo representa el cluster de destino.

Por tanto, los clusters formados en el espacio interno de estados pueden interpretarse como cada uno de los estados que pertenecerían a un autómata finito determinista, ya que las transiciones están completamente definidas y, en todos los casos, son deterministas. No hay ninguna dificultad adicional para generar una tabla de transiciones y extraer directamente el autómata equivalente, tal y como se muestra en la figura 5.5. Este proceso completo de análisis y extracción del autómata se puede aplicar a todos los problemas descritos en la sección 4.1.1. Se pueden encontrar en el apéndice C las tablas de transiciones y el autómata mínimo extraído de cada una de las *Tomita Grammars* [65] descritas en la sección 4.1.1.

	a	b	$
0*	7	5	3
1	6	1	3
2*	7	2	2
3*	7	4	2
4*	7	4	2
5	6	0	2
6	1	1	3
7*	3	5	3

		a	b	$
0	(1, 6)	0	0	4
1	(5)	0	2	4
2*	(0)	3	1	4
3*	(7)	4	1	4
→ 4*	(2, 3, 4)	3	4	4

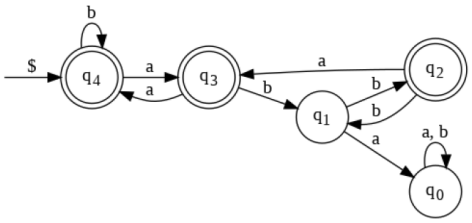


Figura 5.5: Tabla de transiciones de los clusters del problema *Tomita 3* y el autómata finito determinista extraído. El símbolo $*$ identifica un estado final. La tabla izquierda representa la tabla de transiciones original, mientras que la tabla derecha es el resultado de aplicar un algoritmo estándar de minimización de autómatas finitos [26]. El autómata mostrado es el mínimo, extraído de la tabla de la derecha.

5.1.3. Estabilidad

Aunque los resultados presentados en las secciones anteriores son realmente interesantes desde el punto de vista de la interpretabilidad, en el estado del arte los esfuerzos realizados para reducir una red neuronal recurrente a un conjunto de reglas han sido frecuentemente criticados [50] debido a la inestabilidad de los estados extraídos. Si los estados no son verdaderamente estables, una pequeña desviación prolongada en el tiempo puede provocar una divergencia en la activación y, como consecuencia, un comportamiento desconocido y posiblemente erróneo. En esta última sección se realiza un estudio de la estabilidad de este modelo siguiendo dos metodologías diferentes: (1) analizando su comportamiento con cadenas de longitud varias órdenes de magnitud mayor a las de entrenamiento, tal y como se describe en la sección 4.1.1 con el dataset *mega*, cadenas 10^4 veces mayor que entrenamiento, y (2) estudiando el comportamiento del modelo cuando se introducen pequeñas perturbaciones en los estados utilizando ruido Gaussiano.

En primer lugar, en el experimento (1), para analizar la eficiencia del modelo con cadenas extremadamente más largas que las vistas durante el entrenamiento (cadenas con longitud media de 10, aproximadamente), se utiliza el dataset *mega*, 100 cadenas de longitud 10^6. Para cada problema se entrenan 20 redes diferentes con un 100 % de acierto en el conjunto de entrenamiento y se analiza el acierto frente a la longitud de cadena. En este caso, para mostrar una comparación con un modelo estándar en el estado del arte, se realiza el experimento tanto con la Noisy Elman RNN como con una LSTM. Se muestra en la figura 5.6 el porcentaje de acierto frente a la longitud de la cadena en escala logarítmica para los problemas *Tomita 3* y *BxA*. Se añade el problema *BxA* ya que ilustra con bastante claridad el problema, debido a que lo único que debe recordar el modelo a lo largo del tiempo es si la cadena ha comenzado o no con el símbolo b. Se muestran en el apéndice D los resultados de este experimento con cada uno de los problemas regulares sobre el alfabeto $\{a, b\}$ descritos en la sección

4.1.1.

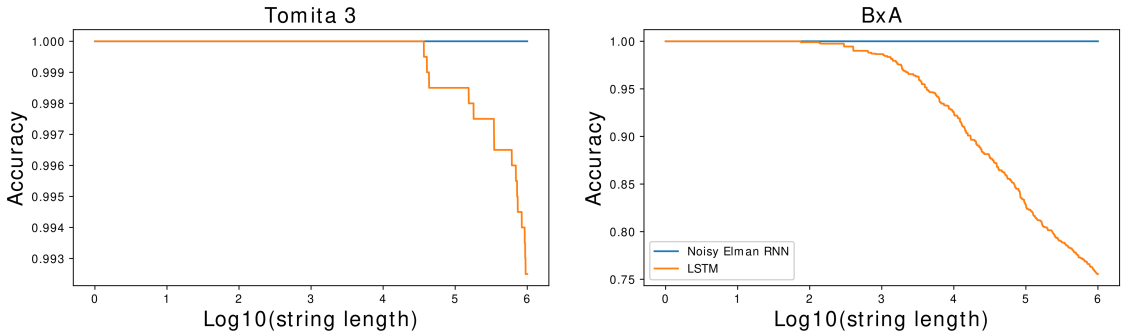


Figura 5.6: Acierto frente a la longitud de cadena para los problemas *Tomita 3* y *BxA* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

La principal observación es que la Noisy Elman RNN mantiene el 100 % de acierto independientemente de la longitud de la cadena, mientras que la LSTM estándar sufre una brusca caída cuando la longitud aumenta varios órdenes de magnitud. El resultado más interesante ocurre en el problema *BxA*, donde a partir de longitud 10^2, que es aproximadamente la longitud de la cadena más larga del conjunto de entrenamiento, la precisión comienza a disminuir en el caso de la LSTM. El resultado obtenido con la Noisy Elman RNN se puede explicar con la formación de los clusters discretos que define la red durante el entrenamiento.

En segundo lugar, después del análisis presentado en la sección anterior, donde se observa que los clusters formados pueden llegar a ser estables, es necesario analizar su comportamiento cuando se introducen pequeñas perturbaciones en dichos clusters. Con este objetivo, en el siguiente experimento se inicializa el modelo en uno de los clusters observados, se inyecta ruido Gaussiano y se mide la activación de las neuronas de la capa recurrente en cada instante de tiempo al procesar una secuencia fija. Se muestra en la figura 5.7 este experimento aplicado para el problema *Tomita 3*. En las cuatro primeras filas de la figura se muestra la salida de las cuatro neuronas activas en 1000 ejecuciones diferentes. La línea roja representa la activación esperada, mientras que los puntos azules representan el valor real. Por otro lado, cada columna representa un instante de tiempo diferente, después de ir procesando cada símbolo de la cadena $\$ab$. La última fila de la figura representa la proyección a dos dimensiones utilizando PCA, donde los puntos naranjas definen el estado actual. De nuevo, los puntos negros representan una expansión Gaussiana de los clusters para que se visualicen con más facilidad.

Como se puede observar en la figura, a pesar del ruido introducido en el estado inicial, la red tiene la suficiente capacidad para recuperar su estado estable y converger a la solución óptima en pocos pasos. Nótese que la inyección de ruido Gaussiano provoca que, aunque no parece suponer ningún problema, en algunas ocasiones, la activación supere el rango válido de las neuronas [-1, 1]. Además, tal y como se muestra en la proyección a dos dimensiones, la perturbación inicial provoca que los 1000 estados tengan una alta dispersión. Sin embargo, prácticamente en dos pasos vuelve a converger al centro del cluster correspondiente. Cabe destacar que la activación siempre converge

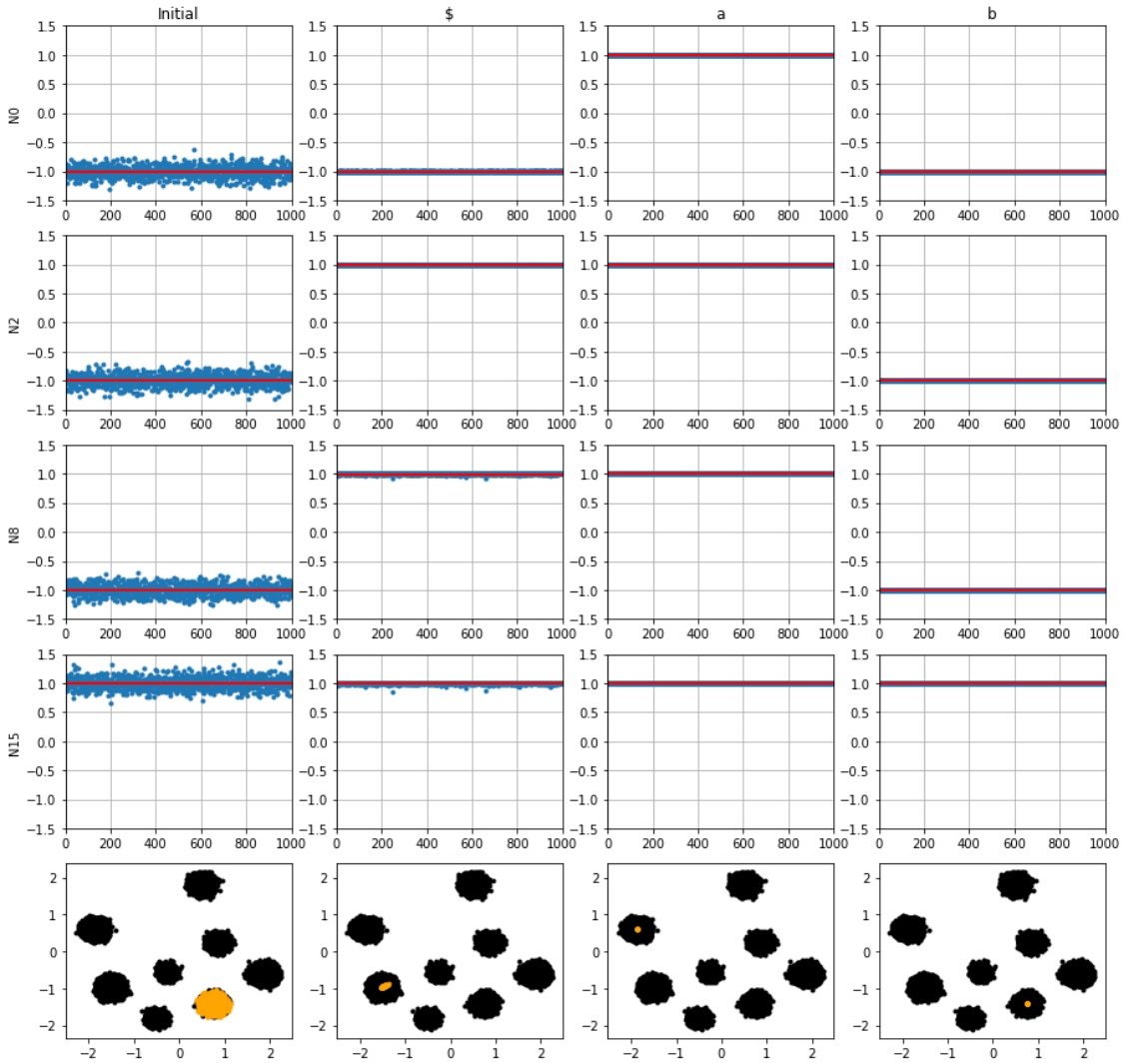


Figura 5.7: Activación en la capa recurrente de una red entrenada con el problema *Tomita 3*. Las cuatro primeras filas representan la salida de cada una de las cuatro neuronas activas para 1000 ejecuciones (perturbaciones) diferentes. Para cada test, el modelo se inicializa en el estado específico que representa la línea roja y se perturba con cierto ruido Gaussiano. La última fila muestra la proyección a 2 dimensiones del espacio de estados interno del modelo utilizando las dos primeras componentes principales utilizando PCA.

al centro del cluster correspondiente independientemente del cluster inicial que es perturbado y del símbolo de entrada. Este comportamiento implica directamente que si en algún caso la activación de la red se viese ligeramente desplazada, el modelo es capaz de estabilizarse y volver al centro del cluster, como si de puntos atractores se tratase. Es decir, el modelo es capaz de transitar de un cluster a otro de forma determinista y estable, tanto a lo largo del tiempo como en el caso en que sufra alguna perturbación.

Resumiendo, las redes entrenadas con ruido en la función de activación son capaces de organizar su espacio de estados interno en un conjunto de clusters discretos y con una actividad completamente estable, lo que les permite generalizar a secuencias con una longitud varias órdenes de magnitud mayor que las cadenas de entrenamiento. Es posible afirmar, pues, que la red es, en estos casos, equivalente al autómata finito determinista que resuelve el problema.

5.2. Análisis del modelo Dual

El modelo Dual RNN surge de la idea de separar completamente el procesamiento de la información temporal, que debe extraerse de la secuencia de entrada, del resto del proceso, donde el modelo debe procesar la respuesta final. Tal y como se ha mencionado en la sección anterior, la idea surge al observar que en el problema de la suma la única información que fluye por la matriz de pesos de la capa recurrente es la respuesta de una única neurona. Además, con esta arquitectura se logra extender el concepto de interpretabilidad y la eficiencia de la Noisy Elman RNN.

Esta sección se divide en los siguientes apartados: en la sección 5.2.1 se muestra, utilizando las Tomita Grammars [65], que el modelo Dual tiene la misma capacidad de generalización, interpretabilidad y estabilidad que la red de Elman con ruido. En la sección 5.2.2 se expone el potencial de la red Dual frente a la Noisy Elman RNN en el problema de la suma en diferentes bases. Por último, en la sección 5.2.3 se hace una comparación entre los dos modelos y la LSTM en términos de eficiencia y, finalmente, se estudia la interpretabilidad del modelo Dual en el problema de la generación de expresiones algebraicas (ver sección 4.1.2).

5.2.1. Primeros resultados: Tomita Grammars

El primer paso para comprobar la eficacia del modelo Dual consiste en estudiar su comportamiento para aprender las *Tomita Grammars* [65] (ver sección 4.1.1). El objetivo es encontrar un resultado equivalente al que se obtendría al resolver cada uno de los problemas con la red Noisy Elman RNN, tal y como se ha mostrado en la sección anterior. No es difícil encontrar parámetros que permitan alcanzar un 100 % de acierto tanto en la fase de entrenamiento como en todos los test realizados. Ambos modelos utilizan los siguientes parámetros: 10 unidades en la capa recurrente y 10 unidades adicionales en

la capa feedforward, en el caso de la red Dual; factor de regularización $r = 0{,}1$ adaptativo (ec. 4.2) con pendiente 4, tamaño del batch de 25, tamaño de desenrollamiento de 25 y, por último, ruido $\nu = 1{,}0$ adaptativo con pendiente 2. Ambos modelos se entrenan utilizando el algoritmo de Adam [69] con un factor de aprendizaje $l = 0{,}01$ durante 1000 épocas.

Después del entrenamiento, en ambos modelos la capa recurrente se ha regularizado en su mayor parte, dejando exclusivamente las neuronas que son necesarias para resolver el problema. Además, las neuronas activas de ambos modelos se han binarizado y, tras realizar el mismo análisis que en la sección anterior, se puede extraer un conjunto finito de estados (clusters) y transiciones entre clusters respondiendo a cada símbolo de entrada, por lo que se puede generar un autómata finito determinista que resuelve el problema. Este comportamiento se puede observar para todas las gramáticas de *Tomita* y los problemas *Paridad* y *BxA*. Se muestra en la figura 5.8 un ejemplo de la activación de las neuronas de la capa recurrente para el problema *Tomita 6* tras procesar 50.000 símbolos de una cadena de test. Como se puede observar, solamente cuatro (las neuronas 3, 4, 5 y 8) de las 10 neuronas tienen una activación completamente binaria a lo largo del tiempo, mientras que el resto se han regularizado (su activación es constante con valor 0).

Sin embargo, es interesante analizar de forma abstracta el autómata obtenido para la red Noisy Elman RNN y compararlo con la red Dual. En ambos casos, después de aplicar el algoritmo de minimización, el autómata obtenido es similar pero no igual. En el caso de la red de Elman, la salida depende exclusivamente del estado de la red, por lo que se puede representar mediante la asociación de un símbolo de salida a cada uno de los clusters del modelo (estados del autómata) y, por tanto, se puede interpretar como una máquina de Moore [30] (ver sección 2.1.1). Por otro lado, la respuesta de la red Dual depende tanto del estado de la capa recurrente como de la información extraída de la entrada, por lo que es necesario introducir el símbolo de salida a la transición del diagrama de estados del autómata correspondiente. En este caso, el diagrama representa directamente una máquina de Mealy [31] (ver sección 2.1.1). Se muestran en la figura 5.9 los autómatas de Moore y de Mealy extraídos de una Noisy Elman RNN y una red Dual, respectivamente, cuando han sido entrenadas con el problema *Tomita 6*, aquellas cadenas cuya diferencia entre el número de a y b es tres.

Aunque los dos autómatas parezcan similares, esta observación es exclusivamente válida para este problema en particular. Es necesario destacar que todas las transiciones que llegan a un estado de la máquina de Mealy están etiquetadas con el mismo símbolo de salida (0 ó 1) y, por tanto, las máquinas de Moore y Mealy tienen exactamente el mismo grafo de transiciones. Sin embargo, este escenario no se cumplirá a medida que se vaya aumentando la complejidad de los problemas de entrenamiento. Cabe mencionar que con todas las *Tomita Grammars* [65], los autómatas de Moore y Mealy extraídos de ambas arquitecturas son equivalentes tanto en forma como en funcionalidad.

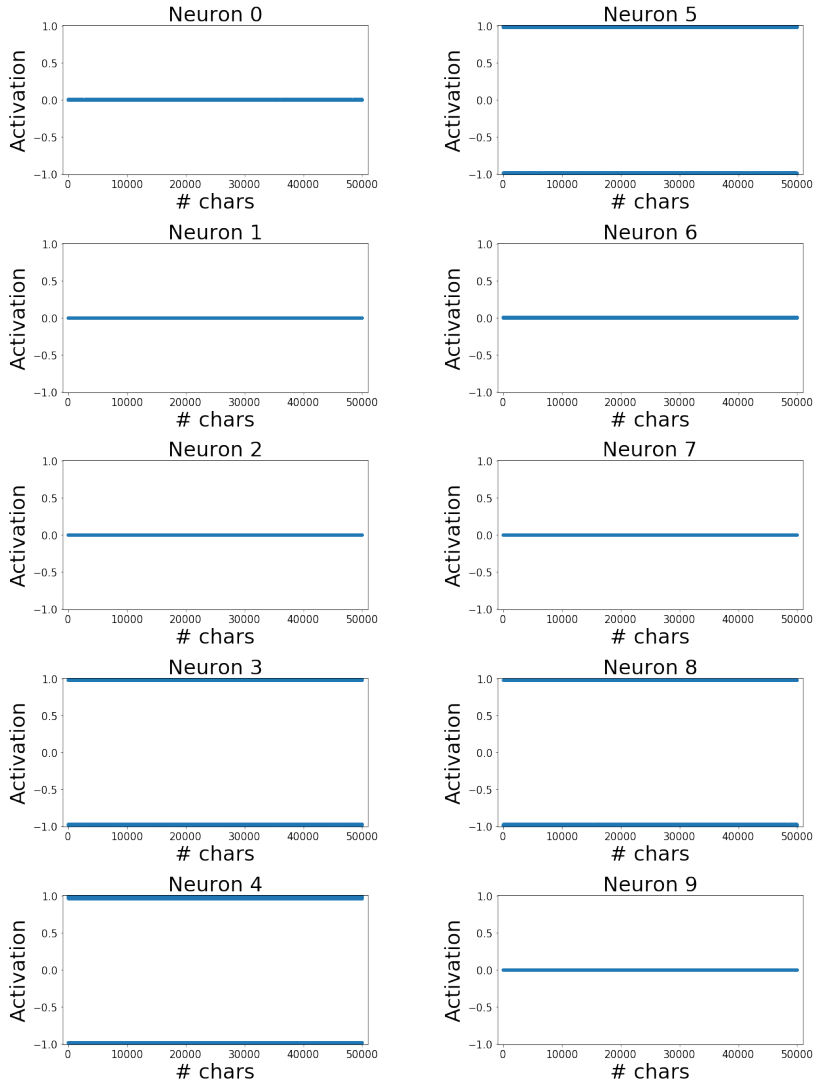
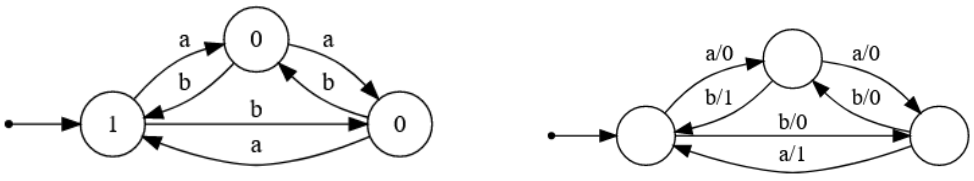


Figura 5.8: Activación de cada una de las neuronas de la capa recurrente frente al tiempo de una red Dual entrenada con el problema *Tomita 6*. Como se puede observar, solamente las neuronas 3, 4, 5 y 8 se encuentran activas y su salida se ha binarizado completamente.



(a) Máquina de Moore (Noisy Elman)

(b) Máquina de Mealy (Dual RNN)

Figura 5.9: Autómatas de Moore y Mealy extraídos de una Noisy Elman RNN y una Dual RNN, respectivamente, entrenadas con el problema *Tomita 6*.

5.2.2. Problema de la suma y su interpretabilidad

El problema de la suma en diferentes bases descrito en la sección 4.1.1 ya ha sido tratado en la sección 5.1 siguiendo un enfoque de análisis y estudio de la estabilidad y la interpretabilidad de la red de Elman con ruido. Sin embargo, se va a volver a estudiar desde otro enfoque diferente, buscando comprender no solo el comportamiento del modelo en su conjunto, sino buscando interpretar el mecanismo individual de cada neurona. Desde este segundo punto de vista se demuestra que el modelo de Elman con ruido tiene una limitación cuanto mayor es la complejidad del problema, que equivale a decir que la base de la suma es mayor. Es entonces cuando surge la red Dual, que aprende a utilizar la memoria de una forma ingeniosa reduciendo considerablemente la complejidad del modelo.

En primer lugar, para los resultados de ejemplo que se van a mostrar a continuación ambos modelos utilizan la misma configuración que en el apartado anterior, exceptuando que en la Noisy Elman RNN se aumenta el número de neuronas de la capa oculta a 20. En todos los resultados, ambos modelos alcanzan un 100 % de acierto en sus respectivos test. De nuevo, los efectos de la regularización, anulando neuronas innecesarias, y del ruido, binarizando las activaciones, se vuelven a observar en este problema. Se muestra a continuación, en la figura 5.10, la activación de las tres neuronas activas de una Noisy Elman RNN entrenada con el problema de la suma en base 2.

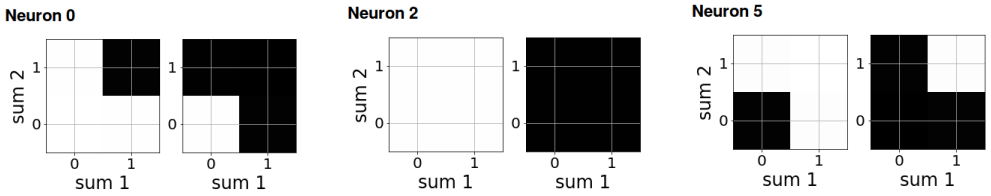


Figura 5.10: Activación de las tres neuronas activas de la capa recurrente de una Noisy Elman RNN entrenada con el problema de la suma en base 2. Para cada neurona se muestran dos figuras: la izquierda representa la activación cuando no hay acarreo, mientras que la figura derecha representa la activación cuando hay acarreo en el instante anterior. Los ejes de cada figura definen el valor de cada sumando.

En este caso, analizando el comportamiento de cada neurona, es posible encontrar una interpretación: la primera neurona (N0) está aprendiendo el acarreo. Nótese cómo, en el caso con acarreo, siempre que uno de los dos sumandos sea 1, se activará la neurona con valor negativo (negro), mientras que, en el caso contrario, solamente cuando ambos sumandos son 1 tiene una activación negativa. Por otro lado, la neurona N2 ha aprendido a mantener el resultado del acarreo en el instante anterior, que será útil para el modelo para proporcionar la salida correcta. Por último, la neurona N5 está resolviendo la no-linealidad de la suma en binario, ya que, en combinación con el acarreo (N0), se comporta como una puerta lógica XOR. Además, es posible nuevamente extraer el autómata finito que resuelve el problema, tal y como se muestra en la figura 5.11.

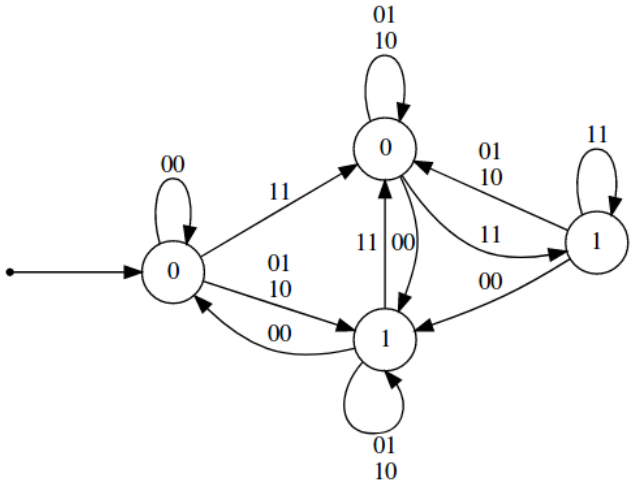


Figura 5.11: Autómata extraído de la RNN de Elman descrita en la figura 5.10 entrenada con el problema de la suma en base 2.

Desde un punto de vista teórico, solamente sería necesario mantener las neuronas N0 y N2 en memoria para resolver el problema de la suma, ya que son las neuronas que mantienen el acarreo. Sin embargo, debido a la necesidad del modelo de mantener la información temporal y, al mismo tiempo, tener la capacidad de procesar la entrada para dar una salida correcta, es decir, sumar, algunas unidades de la capa recurrente se ven forzadas a aprender cierta información que no depende de la historia de la secuencia, sino que solamente depende de la entrada actual. Este inconveniente se ve agravado cuanto mayor es la complejidad del modelo, como es el caso de la suma en base 10. Esta vez, se puede observar un comportamiento similar al ejemplo mostrado en la figura 5.12.

Como se puede observar en la figura, aunque los patrones formados por las neuronas sean no menos interesantes, solamente hay una neurona cuya interpretabilidad es clara: la neurona N1, que está reconociendo el acarreo con activación positiva (blanco). El resto de neuronas codifica de cierta manera la capacidad de procesamiento de la red para dar la respuesta correcta, pero su interpretación no es directa. Una vez más, es posible extraer el autómata finito que resuelve el problema. Sin embargo, en esta ocasión se ha decidido no mostrar el autómata ya que no aporta demasiada información.

Por otro lado, el modelo Dual entrenado bajo las mismas condiciones, con 10 unidades en la capa recurrente y 10 unidades adicionales en la capa feedforward, además de cumplir las observaciones ya realizadas en el caso de la red de Elman, alcanzando un 100 % de acierto en test, solamente tiene dos neuronas de la capa recurrente que mantienen su actividad, quedando el resto regularizadas, tal y como se muestra en la figura 5.13. En este hecho reside la principal diferencia con el modelo Noisy Elman RNN: esas dos neuronas son necesarias para procesar correctamente el acarreo, es decir, la capa recurrente recuerda únicamente lo estrictamente necesario para tratar con la dependencia temporal. Con esa información, la capa feedforward puede procesar completamente la información y proporcionar la respuesta correcta. De hecho, el resultado más interesante consiste en que, independientemente de la base, la red Dual utiliza exclusivamente dos neuronas para el acarreo.

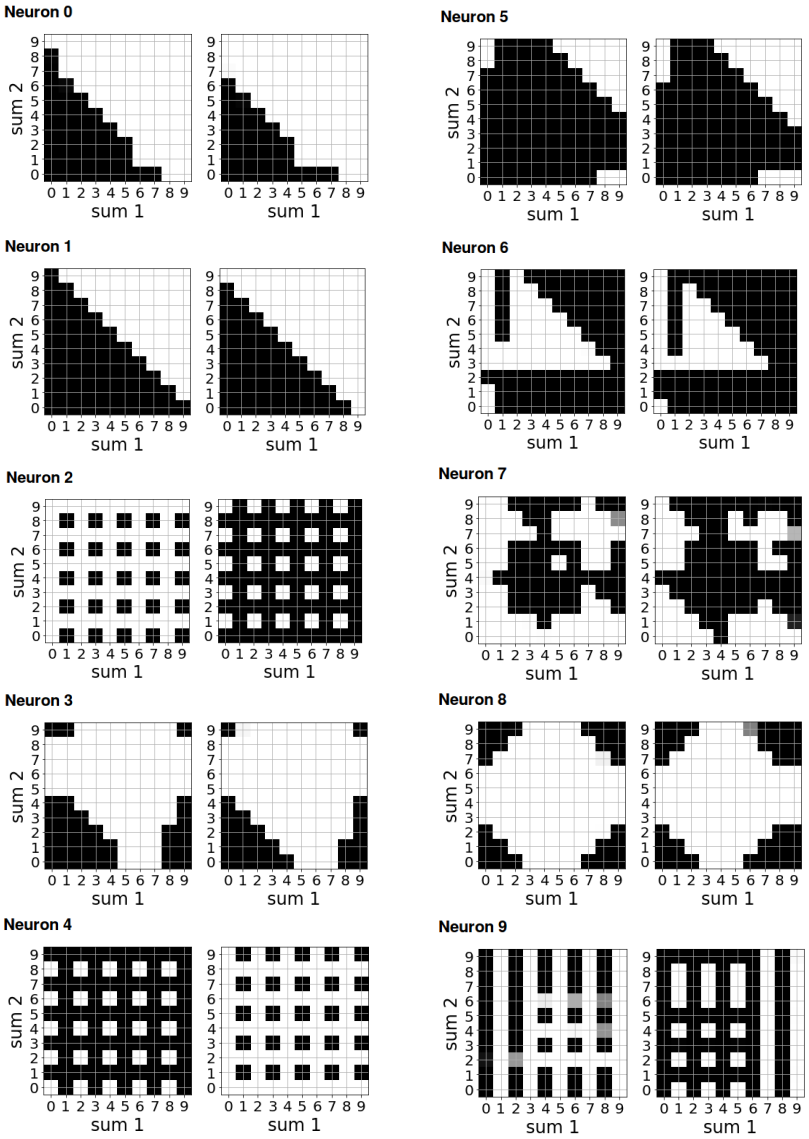


Figura 5.12: Activación de las neuronas activas de la capa recurrente de una Noisy Elman RNN entrenada con el problema de la suma en base 10. Para cada neurona se muestran dos figuras: la izquierda representa la activación cuando no hay acarreo, mientras que la figura derecha representa la existencia de acarreo en el instante anterior.

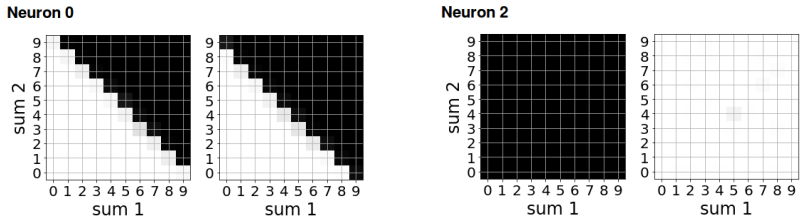


Figura 5.13: Activación de las neuronas activas de la capa recurrente de una Dual RNN entrenada con el problema de la suma en base 10. Para cada neurona se muestran dos figuras: la izquierda representa la activación cuando no hay acarreo, mientras que la figura derecha representa la existencia de acarreo en el instante anterior.

El autómata finito correspondiente se muestra en la figura 5.14. En este caso concreto, se ha extraído de una red Dual entrenada con el problema de la suma en base 2 para que el ejemplo sea más ilustrativo, ya que las etiquetas de las transiciones son más sencillas. Sin embargo, es fundamental comprender que, independientemente de la base, el diagrama de estados es equivalente. Una vez más, el autómata extraído se corresponde a la máquina de Mealy que resuelve el problema, donde los estados se definen mediante la capa recurrente. Este autómata tiene una diferencia fundamental respecto al autómata de Moore: mientras que la máquina de Moore es más compleja cuanto mayor es la base, el autómata de Mealy mantiene la simplicidad de los estados, ya que solo recuerda el acarreo. El estado inicial representa el *no acarreo* y, en el caso de sumar $1 + 1$, transita al otro estado, que sí lo tiene en cuenta. Este comportamiento se puede relacionar directamente con la figura 5.13, donde los dos estados se pueden ver representados en la neurona 2 (activación negra (-1) implica no recordar el acarreo y activación blanca (+1), tenerlo en cuenta).

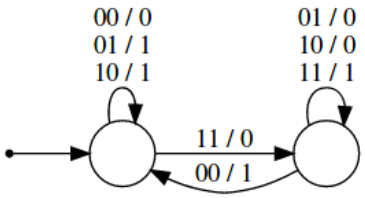


Figura 5.14: Autómata extraído de una red Dual entrenada con la suma en base 2. Es importante mencionar que, independientemente de la base, el autómata mantiene los mismos estados.

Mientras que la Noisy Elman RNN necesita cada vez más capacidad de computación en la capa recurrente y, como consecuencia, necesita aumentar la complejidad de la misma, la red Dual utiliza siempre la misma configuración de memoria, dejando el resto del proceso a la capa feedforward. Resumiendo, la división de la carga de procesamiento en la red Dual permite al modelo centrarse en el procesamiento de la información temporal, simplificando la complejidad de la capa recurrente y su interpretabilidad.

5.2.3. Generación de expresiones algebraicas

Habiendo estudiado el potencial que tiene la red Dual en problemas de clasificación sencillos, donde debe identificar lenguajes regulares, en este experimento se va a probar la capacidad del modelo en problemas de generación de secuencias. Concretamente, el objetivo es ahora generar expresiones algebraicas con una profundidad de paréntesis fija. Con el fin de medir la eficiencia de cada modelo una vez entrenados, se generará una secuencia de 50.000 símbolos. Tal y como se describe en la sección 4.1.2, se dividen las cadenas por el símbolo de escape $ y se mide el porcentaje de cadenas correctamente generadas. En este experimento, se han realizado pruebas con distinto número de unidades en la capa recurrente con las redes de Elman, la LSTM y la red Dual. Se ejecutan 10 modelos diferentes para cada configuración y se selecciona la que menor coste cross-entropy alcanza en va-

lidación. Con el mejor modelo se procede con el resto del experimento para obtener el acierto. Este proceso se repite 10 veces para estimar el acierto promedio de la configuración del modelo.

Los resultados obtenidos con la Noisy Elman RNN se muestran en la tabla 5.1. La primera mitad de la tabla corresponde a una red de Elman vanilla, es decir, sin ruido, mientras que la segunda mitad corresponde a la propia Noisy Elman RNN. Como se puede observar en la tabla, tanto el ruido como la regularización benefician el resultado del modelo cuando se tratan por separado: el ruido permite a la red alcanzar un mayor porcentaje de acierto como máximo; y la regularización permite mejorar considerablemente la varianza provocada por el ruido. Sin embargo, en conjunto, aunque mejora respecto al caso inicial, no se obtienen tan buenos resultados.

Config	Units	Test	Min	Max
noise = 0.0	10	70.5±16.2	51.2	97.3
L1 = 0.0	20	79.0±23.8	39.1	99.0
	30	41.7±7.5	28.9	57.3
noise = 0.0	10	85.6±3.1	80.0	92.0
L1 = 0.1	20	88.2±2.2	83.2	91.3
	30	87.4±3.2	82.0	92.6
noise = 1.0	10	91.8±12.8	60.6	99.0
L1 = 0.0	20	89.9±9.9	67.1	98.9
	30	51.2±24.9	22.4	98.8
noise = 1.0	10	80.7±5.0	71.0	88.2
L1 = 0.1	20	82.8±6.3	75.2	99.2
	30	60.7±13.7	47.0	85.2

Tabla 5.1: Acierto promedio de diferentes Noisy Elman RNN entrenadas para la generación de expresiones algebraicas. La primera mitad de la tabla, con ruido $noise = 0{,}0$, es equivalente a probar una red de Elman vanilla, mientras que la segunda mitad contiene los resultados de la Noisy Elman RNN con ruido $noise = 1{,}0$.

Por otro lado, los resultados obtenidos con la Dual RNN se muestran en la tabla 5.2. Cuando no se introduce ruido en el modelo, los resultados son similares la Noisy Elman RNN, por lo que no aportan ninguna información adicional y se ha decidido no mostrarlos. Sin embargo, cuando se incorpora el ruido, los resultados mejoran considerablemente respecto a la situación inicial. Tal y como se puede observar, cuando no se utiliza la regularización, todas las configuraciones estudiadas alcanzan más de un 99 % de acierto en la prueba, lo cual es un resultado realmente interesante.

Uno de los puntos más interesantes a analizar en este experimento es que la capa recurrente en la red Dual, cuando se introduce ruido, se binariza completamente, tal y como se puede observar en la figura 5.15. En esta figura se muestran cada una de las combinaciones de las activaciones de las neuronas de la capa recurrente. Al ser completamente binarias, se puede afirmar que los patrones

Config	Units	Test	Min	Max
L1 = 0.0	5 - 10	99.2±0.6	97.9	99.8
	5 - 20	99.3±0.5	98.1	99.9
	10 - 10	99.3±0.5	97.9	99.8
	10 - 20	99.3±0.6	97.9	100.0
	15 - 10	99.6±0.3	99.0	99.9
	15 - 20	99.7±0.1	99.5	99.8
L1 = 0.1	5 - 10	86.5±4.9	75.1	94.8
	5 - 20	89.5±3.9	82.8	96.6
	10 - 10	89.5±3.0	85.1	96.4
	10 - 20	90.3±2.7	83.6	92.5
	15 - 10	93.0±2.8	89.4	97.4
	15 - 20	87.7±4.6	79.8	96.1

Tabla 5.2: Acierto promedio de diferentes redes Duales entrenadas para la generación de expresiones algebraicas.

(las combinaciones de las activaciones) que componen los clusters del modelo permiten extraer un autómata de Mealy. Se muestra en la figura 5.16 el autómata de Mealy extraído con aquellos símbolos que tienen una probabilidad mayor que 0.001 de ser generados por la red en cada estado. Es posible que el error observado en promedio se deba a esa probabilidad, leve pero existente, de error en cada instante de tiempo, pues cuando se eliminan estas transiciones de baja probabilidad, el autómata extraído es correcto.

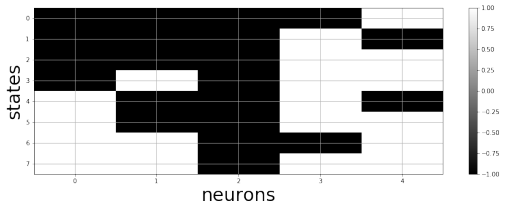


Figura 5.15: Patrones de activación de las neuronas activas de una red Dual entrenada con la generación de expresiones algebraicas con ruido $\nu = 1{,}0$.

Es interesante jugar con el autómata generado y comprobar que es posible generar expresiones algebraicas bien formadas. Parece un juego sencillo a priori, pero tener en cuenta que este autómata representa de forma lógica el comportamiento completo de la red hace que sea un resultado bastante excitante. Además, este mismo proceso de extracción del autómata se puede aplicar a un modelo que obtiene un menor porcentaje de acierto en el test para observar dónde están las transiciones erróneas y, de esta manera, comprender lo que está sucediendo, tal y como se muestra en la figura 5.17, donde se puede ver el autómata erróneo extraído de una red Dual con un 97.38 % de acierto. Con este autómata, si se procesa la sencilla cadena *$a*, hay una probabilidad mayor que 0.001 de aparición del

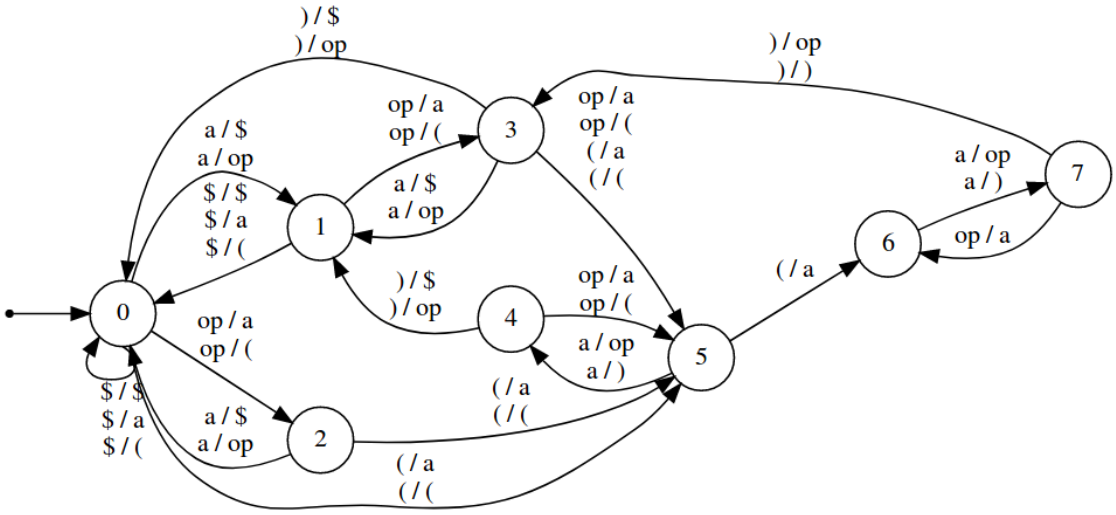


Figura 5.16: Autómata de Mealy extraído de una red Dual entrenada con la generación de expresiones algebraicas que, además, sigue los patrones definidos en la figura 5.15. Solamente se muestran los estados cuya profundidad de paréntesis es menor que 2 para simplificar el grafo, aunque es necesario mencionar que el autómata completo es también correcto. Las transiciones con probabilidad menor que 0.001 también se han omitido.

símbolo), lo que implica un error evidente en la red (la cadena *$a)...$* no debe ser aceptada en ningún caso, ya que no forma parte de la gramática). De hecho, el estado denominado como "1" ni siquiera tiene una transición de salida cuando aparece el cierre de paréntesis, lo cual refuerza el error. Cabe destacar que los fallos detectados en las redes con ruido tienen siempre relación con la apertura y cierre de paréntesis.

Por último, antes de continuar con la siguiente sección, es necesario poner estos resultados en un contexto apropiado, por lo que se muestran en la tabla 5.3 los resultados obtenidos por una LSTM estándar entrenada en las mismas condiciones. La observación principal es que, sorprendentemente, una LSTM estándar no es capaz de alcanzar tanto porcentaje de acierto en el test como lo hace la red Dual. La mejor configuración está 3 puntos porcentuales por debajo del mejor resultado obtenido por la red Dual. Además, la LSTM necesita tanto más unidades como un mayor número de parámetros. Es cierto que el estado del arte actual utiliza variaciones de las redes LSTM estándar que alcanzan mejores resultados, pero la interpretabilidad de la red Dual hace que sea un modelo interesante a tener en cuenta.

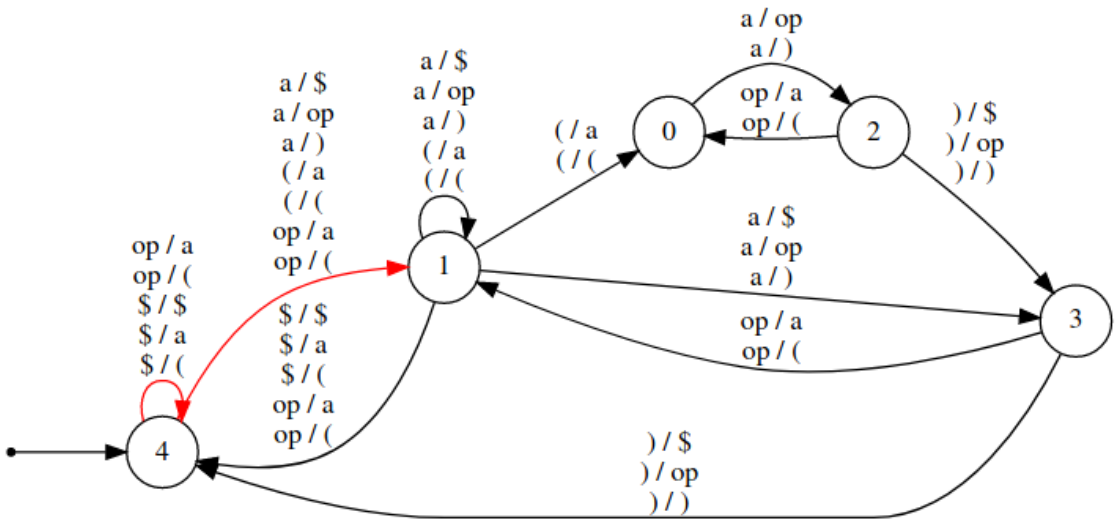


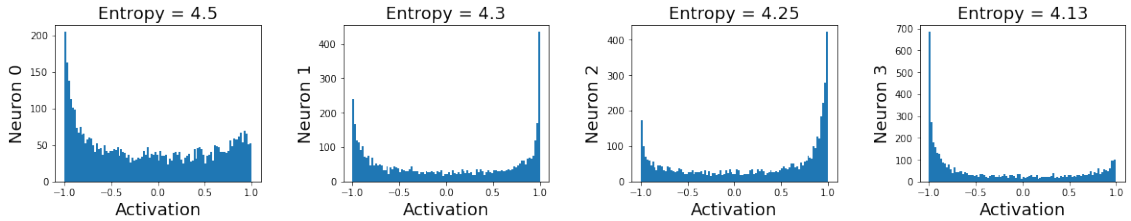
Figura 5.17: Autómata de Mealy extraído de una red Dual que falla con la generación de expresiones algebraicas. Solamente se muestran los estados cuya profundidad de paréntesis es menor que 2 para simplificar el grafo. Las transiciones con probabilidad menor que 0.001 también se han omitido.

Config	Units	Test	Min	Max
L1 = 0.0	10	83.7±3.6	79.4	88.0
	20	88.7±1.1	87.0	90.1
	30	96.5±0.3	96.0	96.9
L1 = 0.1	10	47.3±1.3	45.3	49.2
	20	60.1±0.9	58.9	61.4
	30	64.9±1.0	63.4	66.1

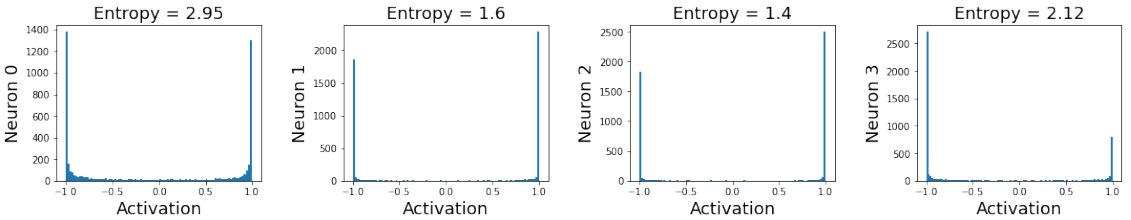
Tabla 5.3: Acierto promedio de diferentes LSTM entrenadas para la generación de expresiones algebraicas.

5.3. Entropía como métrica de interpretabilidad

En las secciones anteriores se ha demostrado que una de las alternativas para conseguir que un modelo tenga una clara interpretabilidad, en forma de un autómata finito, ha consistido en forzar la binarización de la capa recurrente de un modelo mediante la introducción de ruido. En estos casos es posible identificar una serie de clusters que definen de forma lógica el comportamiento de la red. A partir de este punto, donde se va a entrenar el modelo con problemas más complejos, como la generación de texto en español utilizando el Quijote, se ha buscado una métrica que sea capaz de evaluar un "factor de binarización", que asigna una mayor puntuación a las neuronas cuya activación se distribuye en los extremos. Se muestra un ejemplo del objetivo a buscar en la figura 5.18.



(a) Activación con ruido $\nu = 0{,}0$



(b) Activación con ruido $\nu = 1{,}5$

Figura 5.18: Ejemplo de discretización de la activación de las neuronas. En la figura 5.18(a) se muestra la activación de cuatro neuronas seleccionadas al azar de la capa recurrente de una red Dual entrenada con la generación de texto con el Quijote con ruido $\nu = 0{,}0$. En la figura 5.18(b) se muestra la activación para el mismo problema cuando el ruido es $\nu = 1{,}5$.

Como se muestra en la figura, la métrica que se ha decidido utilizar es la entropía de la activación de la capa recurrente. La entropía, que se puede entender como una medida de incertidumbre, disminuye cuanto más discreta sea la activación de las neuronas. Se puede observar en la figura 5.18(a) que el valor de la entropía no disminuye de 4 puntos en cada uno de los histogramas, ya que la activación de cada neurona se distribuye en todo el rango de acción [-1, 1]. Sin embargo, en la figura 5.18(b) se puede observar cómo el valor de la entropía se sitúa, en algunos casos, por debajo de 2 puntos.

Como conclusión, como métrica de interpretabilidad se va a utilizar una aproximación de la entropía de la distribución calculada a partir del histograma de la activación de las neuronas, ya que resulta complicado evaluar un modelo cuando la complejidad del problema comienza a ser demasiado extensa como para estudiarlo manualmente.

5.4. Red Dual en problemas reales

Habiendo introducido la entropía como una métrica de interpretabilidad global de la red y, además, habiendo comprobado que la red Dual tiene un alto potencial, se ha decidido estudiar un problema que es una tendencia en el estado del arte actual: el procesamiento de lenguaje natural (NLP). Concretamente, se va a abordar este campo desde dos perspectivas diferentes: en primer lugar, en la sección 5.4.1 se va a explorar el modelo cuando se entrena con el texto del Quijote para generar texto en español a nivel de carácter y, en segundo lugar, en la sección 5.4.2 se va a utilizar el modelo cuando se enfrenta a un problema de análisis de sentimiento (sentiment analysis), también conocido como minería de opinión (opinion mining).

5.4.1. Generación de texto en español

La generación de texto es uno de los campos de investigación en el procesamiento de lenguaje natural que más auge está teniendo en los últimos años. Existen grandes proyectos multidisciplinares que proponen alternativas e innovaciones con cada vez más potencial, explorando grandes bases de datos, como Wikipedia o PTB [55, 57, 58], entre otras alternativas. En este trabajo de fin de máster se realiza una exploración preliminar del uso de la red Dual en problemas de generación de texto, cuyo objetivo es estudiar los límites del modelo Dual cuando se enfrenta a un problema real.

Una de las incógnitas que han quedado en el aire en la sección anterior, con la generación de expresiones algebraicas, consiste en comprender por qué, cuando se introduce ruido y regularización al mismo tiempo, la red Dual no alcanza tan buenos resultados. Habiendo observado que la mejor configuración era, sin duda alguna, introducir exclusivamente ruido, parece interesante explorar qué sucede en la generación de texto en español. Si resulta que la penalización por regularización es contraproducente frente a la eficiencia del modelo y la mejor alternativa es suprimirla, el principal motivo por el que el ruido se introduce de forma controlada por el estado interno en el tiempo anterior h_{t-1} se desvanece. Es por ello que la primera prueba consiste en analizar dos formas diferentes de introducir ruido en la función de activación: la primera, la forma controlada por h_{t-1} y una segunda en la que se elimina este factor y se introduce ruido Normal sin ninguna modificación.

Con este fin, se van a probar seis configuraciones diferentes con ruido constante en el rango [0.0, 3.0]. Para cada forma de inyección de ruido (dependiente o no de la activación en el paso anterior), se introducen tres configuraciones: (1) la inyección de ruido sin regularización L1, (2) ruido con un factor de regularización $r = 0{,}0005$ constante y (3) ruido con un factor de regularización $r = 0{,}0005$ adaptativo (ver ec. 4.2) con pendiente 2. Cabe destacar que el factor de regularización se ha fijado después de realizar una búsqueda paramétrica con diferentes configuraciones de ruido y número de neuronas. En este experimento se minimiza la función de coste cross-entropy con el optimizador Adam [69] de una red Dual con 200 unidades en la capa recurrente y 500 unidades en la capa feedforward durante

100 épocas. El tamaño del batch se ha fijado a 32, el factor de aprendizaje es 10^{-4} y la longitud de secuencia es 25. En la figura 5.19 se muestran los resultados obtenidos con esta configuración de parámetros.

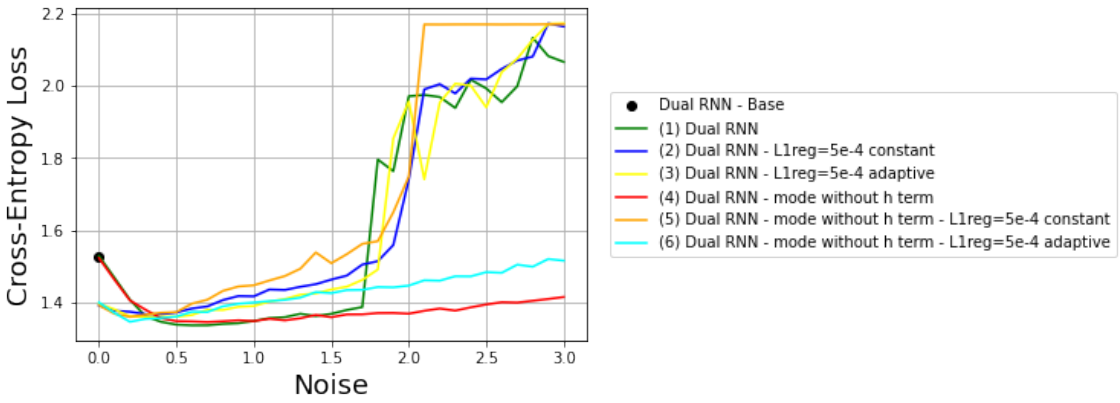


Figura 5.19: Cross-entropy frente a diferentes niveles de ruido en el rango [0.0, 3.0] con las diferentes configuraciones descritas para el problema de generación de texto en español entrenando con el Quijote. El modelo base ($\nu = 0{,}0$ y $r = 0{,}0$) se dibuja con un punto negro para compararlo con las diferentes configuraciones. Cada una de ellas se ha numerado para poder referenciarla en el texto.

Esta figura contiene mucha información que debe ser analizada. En primer lugar, si no se introduce ningún ruido, la regularización L1 (curvas 2, 3, 5 y 6) mejora claramente el modelo estándar (punto negro). Parece lógico, ya que la regularización favorece la reducción de overfitting y la mejora en la generalización. Sin embargo, cuando se aumenta el factor de ruido, los modelos sin regularizar (curvas 1 y 4) mejoran el resultado, lo que implica el conflicto ya mencionado entre el ruido y la regularización.

Por otro lado, analizando las curvas 2 y 5, donde se comparan directamente los dos modos de inyección de ruido, se puede observar cómo la dependencia con el estado interno en el instante anterior favorece a la regularización, como era de esperar. Además, también se puede observar en la figura cómo la regularización adaptativa parece compenetrarse mejor con la inyección de ruido Normal (curva 6) que con la inyección de ruido dependiente (curva 3). Sin embargo, el punto más importante reside en la drástica subida del coste de los modelos cuando el ruido alcanza valores superiores a $1{,}5$, exceptuando las curvas 4 y 6, que mantienen un ligero crecimiento cuanto mayor es el ruido. Este resultado implica que, mientras que la inyección de ruido dependiente alcanza un punto de divergencia, la inyección de ruido independiente del estado previo permite al modelo asimilar una mayor perturbación.

Desde el punto de vista teórico, si la mejor configuración resulta de una inyección de ruido Normal sin ninguna dependencia y con la ausencia de regularización, volviendo la vista atrás a la sección 5.1.1, es posible afirmar que el ruido puede tener una repercusión directa en la interpretabilidad del modelo, ya que provoca una mayor estabilidad. Esta conclusión implica que, con esta configuración, la entropía debe disminuir, tal y como se ha ilustrado en la sección 5.3.

El siguiente análisis, resumido en la figura 5.20, busca encontrar una buena configuración tanto a nivel de eficiencia como de entropía. El objetivo es encontrar los parámetros con los que se pueda alcanzar un coste razonable y una entropía baja. En la figura se muestra la entropía del modelo frente a su cross-entropy para las seis configuraciones anteriores. Sin embargo, aunque el rango de ruido sea [0.0, 3.0], solamente se muestran los puntos cuyo coste no explote y supere el valor $1{,}6$.

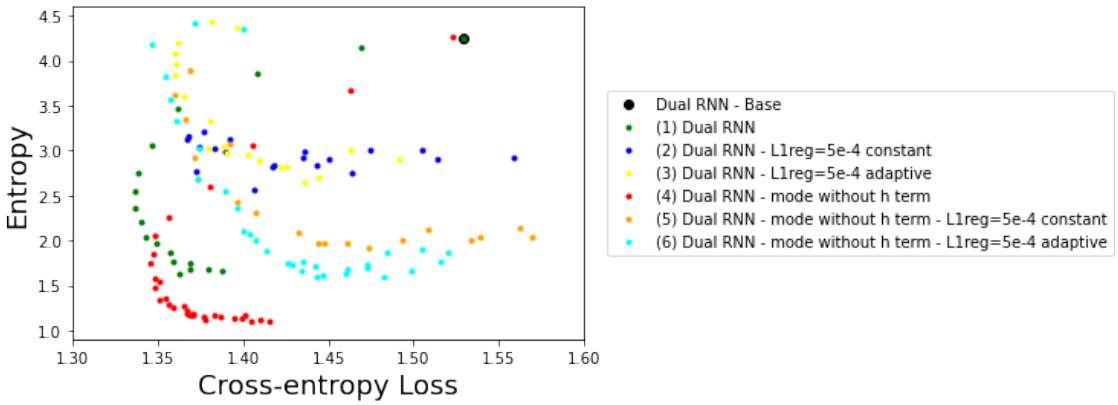


Figura 5.20: Entropía frente al coste cross-entropy para las diferentes configuraciones planteadas del modelo Dual entrenado en la generación de texto utilizando el Quijote durante 100 épocas. El modelo base ($\nu = 0{,}0$ y $r = 0{,}0$) se dibuja con un punto negro para comparar. Las diferentes configuraciones se han numerado para que sea más sencillo citarlas en el texto.

La respuesta de este experimento es seleccionar una configuración que en la figura aparezca en la esquina inferior izquierda, donde hay tanto un cross-entropy como una entropía bajos. Por lo tanto, la configuración (4), es decir, la inyección de ruido independiente del estado anterior sin regularización, con un nivel de ruido en torno a $\nu = 2{,}0$, es la opción seleccionada. En conclusión, introducir una cantidad moderada de ruido Normal sin regularización no solo reduce la entropía, sino que mejora el overfitting y la capacidad de generalización.

Interpretabilidad

A partir de la configuración anterior, el siguiente paso es realizar un análisis del espacio de estados interno de la red Dual para ver si es posible interpretarlo. Todos los resultados mostrados en esta sección se han obtenido de una red Dual con ruido $\nu = 2{,}0$ con 200 unidades en la capa recurrente y 500 unidades en la capa feedforward. Esta red obtiene un coste cross-entropy de $1{,}39$ y $1{,}04$ de entropía promedio. El primer experimento consiste en extraer algunos estados discretos, donde un estado se define como el vector de activación de la capa recurrente de la red en respuesta a cada uno de los símbolos de entrada. Para ello se utiliza el algoritmo Isomap sobre los últimos 30.000 estados del conjunto de datos de validación para transformar un espacio de 200 dimensiones a datos en 2D y, una vez transformados, se seleccionan los 5.369 estados que corresponden a los caracteres de fin de palabra. En la figura 5.21 se dibuja la transformación Isomap de los estados, en la que se han seleccionado de forma aleatoria los estados que se etiquetan con la palabra que representan.

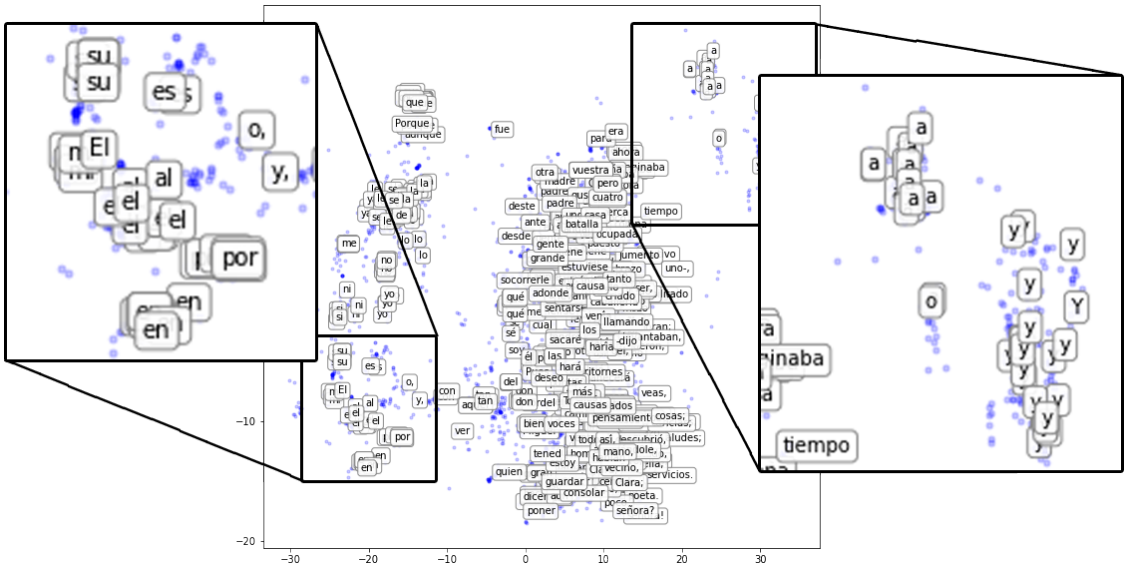


Figura 5.21: Transformación Isomap de los 5369 estados de la red Dual que corresponden a los caracteres final de palabra. Las etiquetas se han dibujado de forma aleatoria. Dos áreas se han aumentado para mostrar palabras similares.

Es una aproximación inicial bastante interesante, donde palabras aparentemente similares parecen estar representadas con estados también similares. Sin embargo, el cluster masivo en el centro de la figura agrupa todas aquellas palabras que tienen baja frecuencia. Este comportamiento podría tener sentido ya que el algoritmo Isomap utiliza internamente el algoritmo K-vecinos próximos utilizando la distancia Euclídea, por lo que esas palabras con baja frecuencia no tienen vecinos próximos para formar una agrupación diferente y alejarse del cluster masivo central. Cabe destacar que la novela de *Don Quijote* tiene más de 25.000 palabras diferentes, por lo que, a priori, parece natural que esto suceda.

La similitud de palabras representada por estados próximos ha conducido a analizar palabras concretas y bien conocidas de la novela, como, por ejemplo, *Quijote*, *Sancho* o *caballero*, entre otras. Se esperaría que los estados que representan la misma palabra fuesen prácticamente iguales, a la vez que mantuviesen cierta distancia con el resto. En primer lugar, se muestra en la figura 5.22 una matriz en escala de grises que representa los 46 estados que corresponden a la palabra *Quijote*.

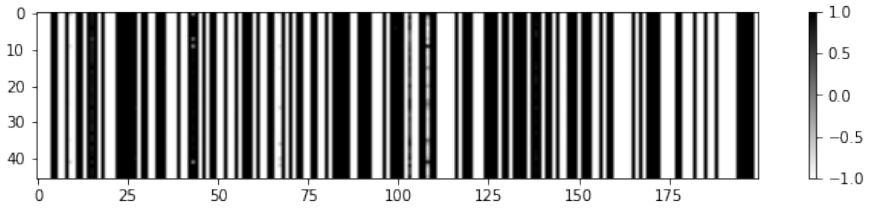
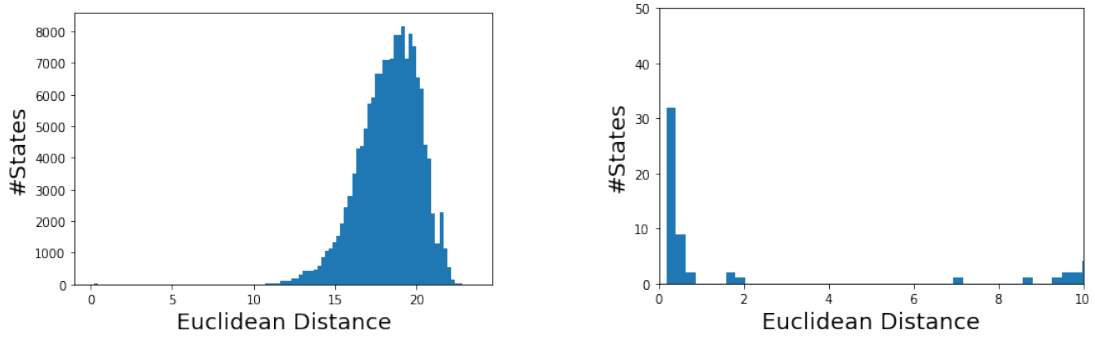


Figura 5.22: Matriz de estados que corresponde a los 46 estados que representan la palabra *Quijote*.

Esta matriz proporciona cierta información a tener en cuenta: en primer lugar, la activación de las neuronas es prácticamente binaria y, en segundo lugar, exceptuando algunas pequeñas variaciones, los 46 estados son prácticamente iguales, por lo que la figura tiene la forma de un código de barras. Se puede afirmar que el promedio de estos 46 estados es una buena aproximación de la palabra *Quijote*. Esta observación aplica a todas las palabras que se han probado de forma individual, por lo que podría afirmarse que es posible asignar cada una de las palabras a un código de barras, lo cual es un resultado interesante.

El segundo paso del experimento es, una vez se ha comprobado que una misma palabra repite el mismo patrón, medir la diferencia entre el vector promedio que representa una palabra concreta y el resto de estados del conjunto de validación, ya sean estados final de palabra o no, utilizando la distancia euclídea. Se muestra en la figura 5.23 un histograma de las distancias entre la palabra *Quijote* y el resto de estados del conjunto de validación.



(a) Histograma completo

(b) Histograma ampliado

Figura 5.23: Histograma de la distancia euclídea entre el vector que representa la palabra *Quijote* y el resto de estados pertenecientes al conjunto de validación. En la figura 5.23(a) se muestra el histograma completo, mientras que en la figura 5.23(b) se ha ampliado la cola izquierda.

Esta figura nos proporciona una importante información, que además se puede aplicar a todas las palabras que se han probado. Como se puede ver en la figura 5.23(b), hay un cierto número de estados que tienen una distancia menor que 2, que corresponden a los 46 estados "*Quijote*", mientras que todos los demás tienen una distancia mayor que aproximadamente 10 (ver figura 5.23(a)).

El siguiente experimento consiste en probar la robustez de la red Dual frente a cierta perturbación, por lo que se va a proceder de la siguiente manera: (1) se resetea el estado interno del modelo a su valor inicial y se procesa una palabra aleatoria. (2) Después de procesar la palabra completa, se añade cierto ruido Gaussiano a la pre-activación de la capa recurrente y es entonces cuando se aplica la función de activación *tanh*. (3) Una vez se ha perturbado el estado interno, se calcula la distancia euclídea de ese nuevo estado perturbado al resto de estados del conjunto completo de validación. (4) Por último, se mide la precisión cuando se recuperan los 10 estados más cercanos al estado perturbado (P@10). Se muestra en la figura 5.24 la precisión promedio frente al nivel de perturbación para dos

redes Duales, una sin ruido y otra con ruido $\nu = 2{,}0$ cuando se prueba con las siguientes palabras: *Quijote* (46), *Dorotea* (25), *Zoraida* (61), *caballero* (20), *cristianos* (37), *tiempo* (30) y *merced* (27). El número entre paréntesis indica el número de apariciones de la palabra en el conjunto de validación.

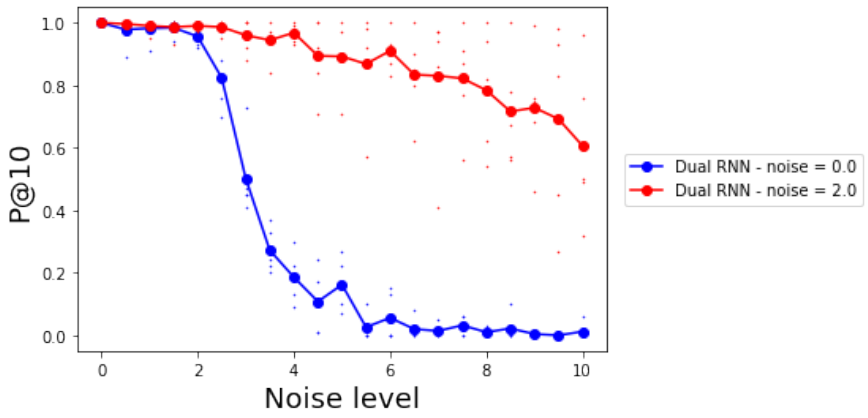
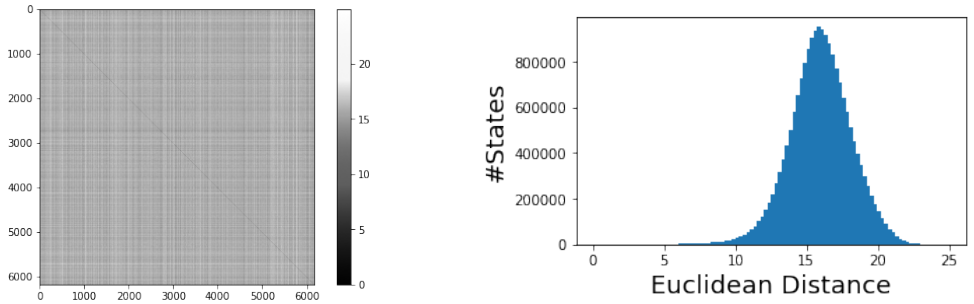


Figura 5.24: Precisión P@10 promedio frente al nivel de ruido de perturbación cuando se prueban las palabras mencionadas en el texto.

Como era de esperar, la inyección de ruido Normal durante el entrenamiento es un mecanismo interesante para mejorar la robustez del modelo frente a ligeras perturbaciones de ruido Gaussiano en el estado de la red. Como se puede observar en la figura, en el caso con ruido ($\nu = 2{,}0$) el modelo es capaz de mantener un 80 % de precisión incluso con un nivel de ruido 3 veces mayor que el empleado durante el entrenamiento.

En el último experimento se ha analizado el espacio de estados interno de validación en su totalidad. Se ha medido la distancia euclídea entre cada par de estados promedio diferentes, donde un estado promedio es el vector promedio de todos los estados internos que representan la misma palabra. Se muestra el resultado obtenido en la figura 5.25, donde se muestra la matriz de distancias entre estados y la distribución que sigue dicha distancia.



(a) Matriz de distancias

(b) Distribución de distancias

Figura 5.25: En la figura 5.25(a) se muestra la matriz de distancias entre pares de palabras en escala de grises. En la figura 5.25(b) se muestra la distribución que sigue la distancia entre palabras.

La causa de que la matriz de distancias se vea prácticamente con el mismo tono se explica con que las distancias siguen una distribución Gaussiana centrada en torno a 15. Este resultado implica que las palabras están lo suficientemente distantes del resto, por lo que podría resultar sencillo identificar en qué situación se encuentra el modelo, de forma global, si buscamos el estado más próximo al real. Sin embargo, en la cola izquierda de la distribución de distancias se ha podido observar que aquellos pares de estados que tienen una distancia menor que 3 tienen una peculiaridad que los hace especialmente interesantes: son palabras sintácticamente equivalentes, tal y como se muestra en la tabla 5.4.

Word	**Similar to...**
movieron	pusieron, hubieron, subieron, tuvieron, ofrecieron, repartieron...
felicísimo	invictísimo, alegrísimo, grandísimo, hermosísimo, serenísimo...
entendimiento	atrevimiento, advertimiento, arrepentimiento, entretenmiento...
tomándome	rogándome, dejándome, mandándome, llevándome, echándome

Tabla 5.4: Ejemplos de palabras cuya distancia euclídea es menor que 3. Como se puede observar, son sintáctica y morfológicamente equivalentes

Por ejemplo, *movieron* y sus vecinos próximos son la tercera persona del plural del pretérito perfecto simple de diferentes verbos, o *felicísimo* y sus vecinos son la forma superlativa de algunos adjetivos. Con todos estos experimentos se ha mostrado que una misma palabra en diferentes contextos se identifica con un vector muy similar en todos los casos y, además, se diferencia del resto de palabras de forma considerable, excepto aquellas que son sintácticamente equivalentes. Este resultado demuestra que el modelo Dual es una herramienta capaz de diferenciar palabras en el conjunto de datos e incluso estructuras sintácticas, al menos identificando la parte final, demostrando cierta capacidad de generalización e interpretabilidad.

5.4.2. Análisis de sentimiento

El análisis de sentimiento, también conocido como minería de opinión, es una disciplina del procesamiento de lenguaje natural caracterizada por identificar o categorizar un texto, review u opinión según la actitud o el sentimiento del escritor. Este enfoque se puede realizar de dos maneras diferentes: (1) un análisis discreto en el que, indicando un umbral, la opinión puede ser positiva, negativa o neutral y (2) un análisis continuo en el que el modelo debe aproximar la puntuación que el usuario haya proporcionado. Básicamente estas dos alternativas describen la diferencia entre un problema de clasificación y un problema de regresión, los dos paradigmas clásicos del aprendizaje supervisado.

Este campo de investigación ha tenido un gran auge en los últimos años, provocado, en mayor parte, por el crecimiento masivo de las redes sociales y el interés por analizar los comentarios en la mayoría de disciplinas que uno pueda imaginar: transacciones bancarias, ciencia, etc., lo que ha llevado a la minería de datos a combinar técnicas clásicas de recuperación de información con el

aprendizaje automático y el procesamiento del lenguaje natural. Así pues, se va a aplicar la red Dual en el área del análisis de sentimiento de las opiniones de los usuarios extraídas de la plataforma de películas de IMDB [64]. El conjunto de datos que se va a utilizar se describe en la sección 4.1.4.

Preprocesamiento de los datos

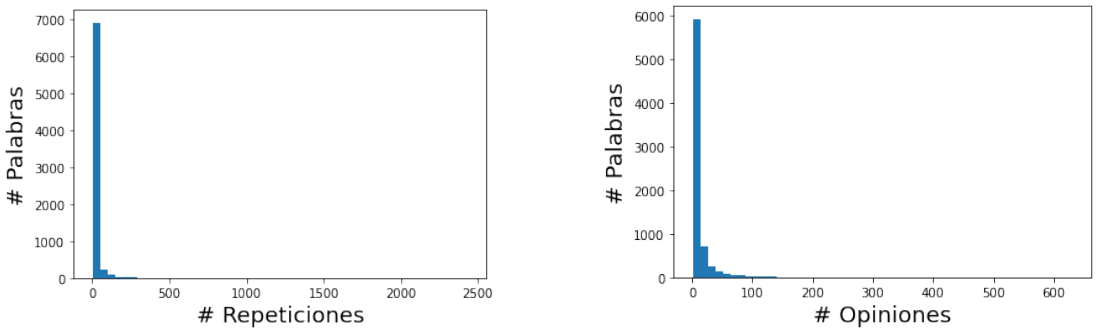
El primer paso para aplicar análisis de sentimiento pasa por realizar una fase de preprocesamiento de los datos en crudo. Estos datos son, literalmente, las páginas HTML de opiniones que pertenecen a cada una de las películas visitadas por el crawler. El preprocesamiento ha consistido en lo siguiente: en primer lugar, extraer las primeras 25 opiniones de cada página HTML, ya que una de las páginas podía llegar a tener más de 1.000 opiniones y se tomó la decisión de acortarlo, es decir, se extrae el texto en crudo y su puntuación en el rango [0, 10]. Con este procedimiento se tienen 25 opiniones de cada una de las 46 películas visitadas por el crawler en la fase de test. Este primer paso favorece que términos concretos, como nombres propios o cualquier suceso puntual de una película en particular, no puedan determinar, a priori, la puntuación.

En segundo lugar, procede la eliminación de símbolos que no interesan, tales como símbolos de exclamación, paréntesis, comillas, etc., una técnica bastante común para simplificar el proceso. De igual manera, se eliminan *stopwords*, palabras comunes de un lenguaje, tales como preposiciones, artículos, etc. y se lematizan las palabras, un proceso lingüístico que consiste en hallar el lema correspondiente de una palabra flexionada, es decir, plural, formas verbales, etc. En la tabla 5.5 se muestran algunos ejemplos de eliminación de símbolos, *stopwords* y la lematización de algunas palabras.

Técnica	**Ejemplos**
Eliminación de símbolos	Hello! $\rightarrow$ hello
	He said: The Lord of The Ring $\rightarrow$ he said the lord of the ring
Eliminación de *stopwords*	between, yourself, but, an, some, for, do, is, am, who, me, were, her...
Lematización	happens $\rightarrow$ happen
	talkative $\rightarrow$ talk

Tabla 5.5: Ejemplos de eliminación de símbolos, *stopwords* y la lematización de algunos términos.

A continuación, se ha tomado la decisión de eliminar tanto las palabras con alta frecuencia como las palabras que solamente aparecen en un documento ya que, como se muestra en la figura 5.26, existen algunas palabras con mucha frecuencia de aparición (la cola de la figura 5.26(a)) y, además, existen muchas palabras que aparecen en pocas opiniones (el pico de la figura 5.26(b)). Esta decisión surge de la necesidad de eliminar palabras comunes en temática de cine, como son *movie*, *film* o *one*, o el caso contrario, eliminar palabras demasiado específicas, tales como *baggins* o *hobbits*, que, claramente, hacen referencia a alguna película de *El Señor de los Anillos* o *El Hobbit*.



(a) Frecuencia de términos

(b) Aparición en opiniones

Figura 5.26: En la figura 5.26(a) se muestra el número de palabras frente al número de repeticiones de cada término, lo cual es equivalente a la frecuencia de aparición de cada palabra. En la figura 5.26(b) se muestra el número de palabras frente al número de opiniones en las que dicho término aparece.

Después de aplicar todo el preprocesamiento, los resultados que se muestran a continuación se han obtenido de un dataset donde hay 839 opiniones, de las cuales la más larga tiene 606 términos. Por tanto, el siguiente y último paso consiste en ejecutar un padding por la izquierda, rellenando con ceros, para que todas las cadenas tengan la misma longitud. El conjunto de entrenamiento contendrá un 80 % de las opiniones, quedando el 20 % restante para la validación.

Resultados

Todos los experimentos planteados previamente tienen en común que se plantea una arquitectura de entrenamiento Many-to-Many, es decir, una arquitectura donde, en cada instante de tiempo, la red proporciona una salida estimada. Sin embargo, en este problema en particular, la arquitectura considerada se basa en el esquema Many-to-One, donde es necesario procesar una serie temporal completa, o en este caso un texto, para dar la estimación final. Se muestra en la figura 5.27 un diagrama con los esquemas típicos en el entrenamiento de los modelos de redes neuronales recurrentes.

En este último experimento se va a realizar una comparación de la eficiencia del modelo Dual frente a un modelo que se puede considerar como estándar: una LSTM con 100 unidades en la capa recurrente cuya entrada ha sido previamente procesada por una capa de Embedding con otras 100 unidades. Además, se utiliza la técnica de dropout con un factor de 0.4. En ambos modelos, el tamaño del batch se ha fijado a 10. Se muestra en la figura 5.28 el diagrama generado por Keras para describir el modelo final que define el modelo Dual. Esta red procesa el texto en formato OneHot (la función *lambda*) a diferencia de la LSTM, que procesa el texto utilizando la capa de Embedding.

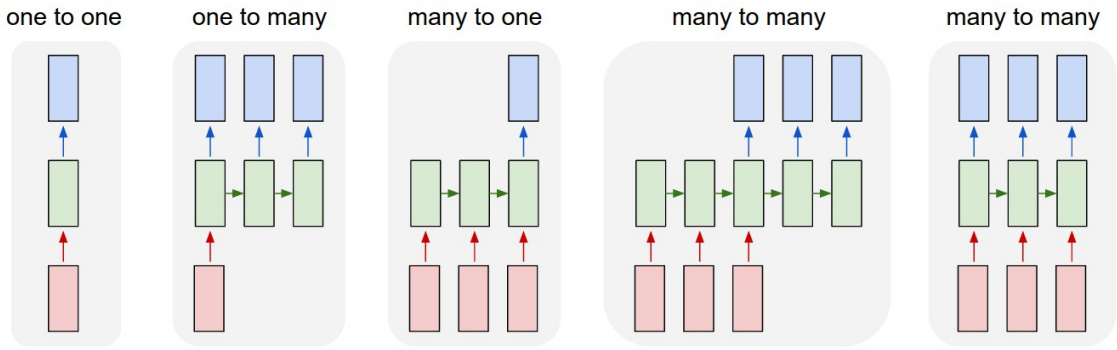


Figura 5.27: Diagrama donde se resumen las principales arquitecturas en el entrenamiento de los modelos de redes neuronales recurrentes. Imagen extraída de http://karpathy.github.io/2015/05/21/rnn-effectiveness/

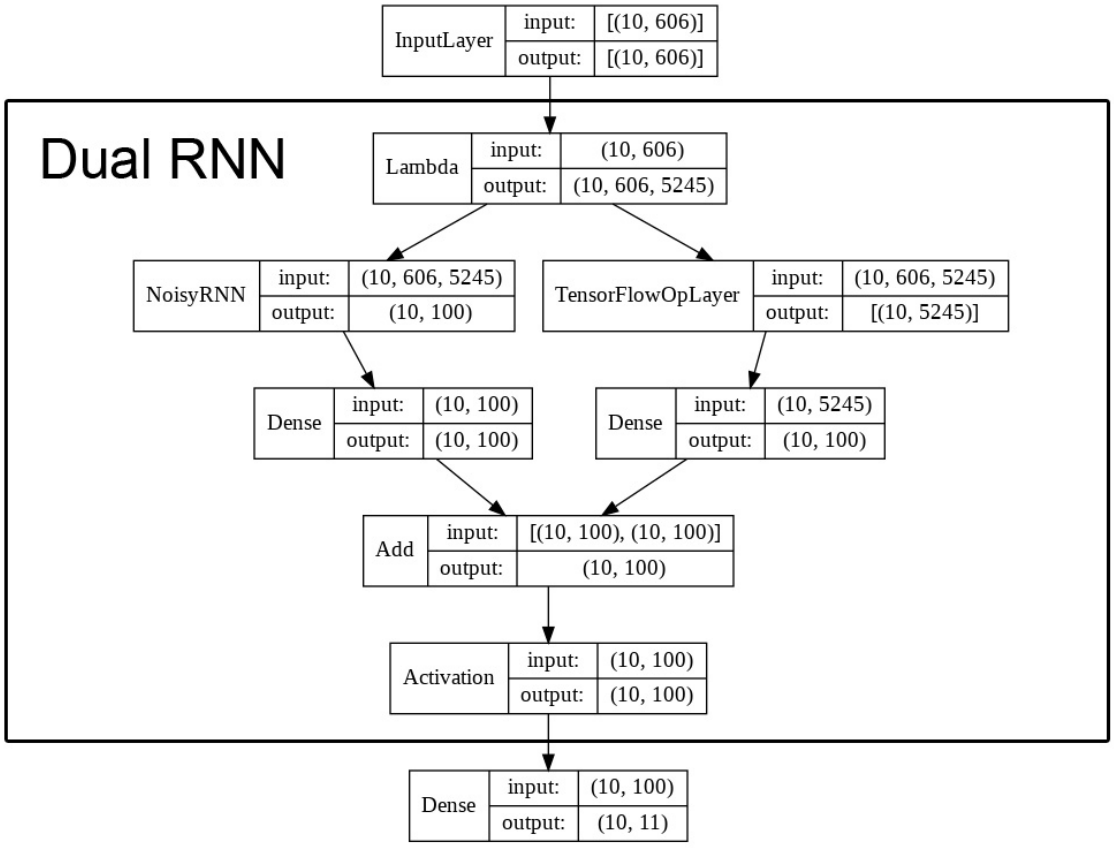
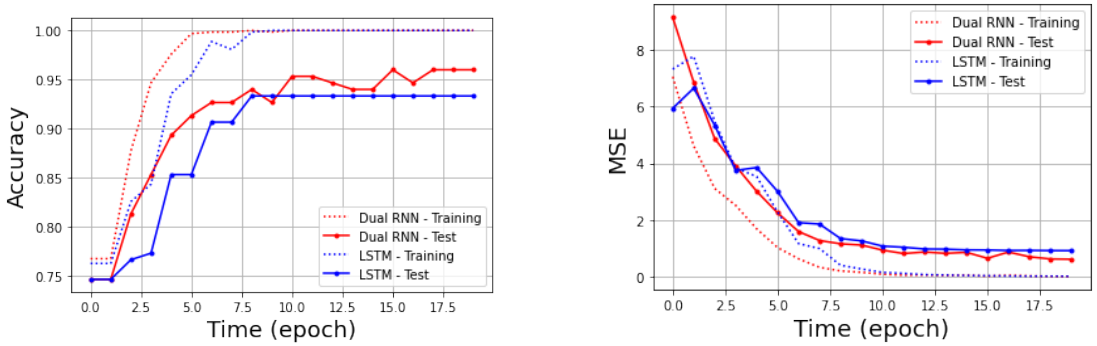


Figura 5.28: Diagrama del modelo de la red Dual generado por Keras aplicado al problema de análisis de sentimiento. En este diagrama se puede ver perfectamente el concepto de dualidad, donde la entrada se divide en dos ramas diferentes. En esta figura, la capa Lambda define una transformación de la entrada a formato OneHot (5245 términos, de ahí su dimensión de salida) y la capa TensorFlowOpLayer selecciona únicamente el último término del texto para aplicarlo a la capa feedforward. Cabe destacar que la capa feedforward no tiene bias, ya que es necesario aplicar un único bias a la suma y la consecuente función de activación.

En el diagrama se puede observar cómo, en primer lugar, se aplica una función Lambda a la entrada para transformarla a formato OneHot. A partir de este punto, la red se divide en la parte recurrente (la rama izquierda) donde se aplica el modelo *NoisyRNN*, una modificación de la capa *SimpleRNN* de Keras [22] donde se introduce el ruido en la función de activación, tal y como se describe en la sección 4.2.2. Por el otro lado, la rama feedforward (derecha) se aplica exclusivamente al último término del texto (la capa *TensorFlowOpLayer*) para, por último, aplicar la función de activación a la suma de las dos partes. Antes de entrar en detalle con los resultados, cabe destacar el pensamiento adicional que condujo a utilizar un aprendizaje por clasificación minimizando la función cross-entropy y no por regresión, como sería natural a priori en un problema donde hay que aproximar una puntuación en el rango [0, 10]. Sin embargo, se ha comprobado que también se reduce el error cuadrático medio (SME) de forma considerable. Si se diese el caso de que la predicción fuese drásticamente diferente del valor real, que podría suceder al entrenar utilizando cross-entropy, la métrica SME lo identificaría.

Por último, los resultados que se muestran se han obtenido aplicando los siguientes parámetros: el número de neuronas en la capa recurrente, si hablamos tanto del modelo Dual como del modelo LSTM, es 100. El número de neuronas en la capa feedforward en el caso de la red Dual, o en la capa de Embedding en el caso de la LSTM, es 100. El tamaño del batch se ha fijado a 10. El tamaño de secuencia o desenrollamiento es la longitud de la opinión más larga, es decir, 606. Ambos modelos se entrenan durante 20 épocas utilizando el optimizador Adam [69] con un factor de aprendizaje de 10^{-4}. Es importante destacar que la red Dual tiene el factor de ruido $\nu = 0{,}5$. Ahora sí, se muestran en la figura 5.29 dos métricas de evaluación de los modelos, en la figura 5.29(a) se muestra el acierto frente al tiempo y en la figura 5.29(b) se muestra el error cuadrático medio frente al tiempo.



(a) Acierto frente al tiempo

(b) MSE frente al tiempo

Figura 5.29: En la figura 5.29(a) se muestra el acierto frente al tiempo de la red Dual y la LSTM. En la figura 5.29(b) se muestra el error cuadrático medio frente al tiempo.

Como se puede observar en las figuras, aunque ambos modelos tienen una eficiencia de más de un 90 %, es cierto que el modelo Dual mejora ligeramente a la LSTM, alcanzando un 96 % en validación, mientras que la LSTM se queda en un 94 %. Además, tal y como se ha mencionado anteriormente, el valor MSE demuestra que aquellos fallos en validación no se corresponden a decisiones dispares.

Es necesario tener en cuenta que el valor MSE se ha calculado promediando sobre 1000 iteraciones por cada modelo en cada época, por lo que puede considerarse como un valor real que no depende del azar provocado por la selección probabilista de cada puntuación. Cabe destacar también que la red Dual, cuando no recibe ruido, no es capaz de superar el acierto de la LSTM, por lo que el ruido favorece, una vez más, a la capacidad de generalización y, por tanto, la eficacia del modelo.

6

Conclusión

El objetivo de este trabajo ha consistido en realizar un estudio de la equivalencia entre las redes neuronales recurrentes y los autómatas finitos deterministas desde dos puntos de vista diferentes: en primer lugar, se ha realizado un análisis empírico de la estabilidad y la capacidad de generalización de la red de Elman modificada con ruido Gaussiano en la función de activación y, por otro lado, se ha hecho un estudio completo de la nueva arquitectura de red Dual. Siguiendo estos objetivos, se han presentado en el capítulo 2 los conceptos básicos que deben conocerse para comprender completamente el contenido de este trabajo. En el capítulo 3 se ha realizado un estudio del estado de la cuestión sobre el aprendizaje automático y el procesamiento de lenguaje natural. En el capítulo 4 se ha presentado el diseño de los problemas que se han estudiado a lo largo del proyecto, además de las dos arquitecturas de red desarrolladas e implementadas. Por último, en el capítulo 5 se han presentado todos los resultados, con sus respectivas pruebas y evidencias.

Respecto a la estabilidad de la red de Elman modificada cuando es entrenada con cierta cantidad de ruido con problemas de identificación de lenguajes regulares, tales como las *Tomita Grammars* [65] o la suma de dos sumandos en diferentes bases, se ha demostrado que las redes entrenadas son capaces de organizar el espacio de estados interno en un conjunto de clusters discretos y con una actividad completamente estable, ya que son capaces de recuperarse de una perturbación momentánea. Este resultado permite a estas redes tener una alta capacidad de generalización y, gracias a ello, se ha demostrado que la red es equivalente al autómata finito determinista que resuelve el problema.

Por otro lado, el desarrollo de la red Dual ha resultado ser altamente satisfactorio. Por un lado, tiene la misma capacidad de generalización y estabilidad que la red de Elman modificada cuando se enfrenta a los problemas regulares antes mencionados. Además, su arquitectura permite que la información puramente temporal sea procesada por separado del resto de la información de los datos del problema, por lo que se simplifica notoriamente la complejidad de la capa recurrente y su interpretabilidad. De esta manera, la interpretabilidad del modelo queda reducida a analizar la capa recurrente, que actúa como una memoria externa. Ya que los estados internos son estables y discretos y, además, la salida de la red depende tanto de la entrada como de su estado interno en un instante determinado, este modelo es directamente equivalente a una máquina de Mealy.

Estos resultados demuestran que una red neuronal recurrente puede implementar un autómata finito determinista cuando se entrena para identificar lenguajes regulares, ya sea en la forma de una máquina de Moore o de Mealy, lo cual da una interpretación directa del modelo completo de la red neuronal. Por otra parte, la idea de separar la capa recurrente del resto del proceso parece tener un futuro prometedor incluso en problemas del mundo real. En los experimentos realizados, la red Dual con ruido mejora considerablemente la entropía de la activación de la capa recurrente, métrica que se utiliza como indicador de la discretización de la red y, por tanto, de su interpretabilidad como autómata. Además, también mejora la capacidad de generalización y por tanto su eficacia, llegando incluso, en algunos casos, a mejorar los resultados obtenidos con una LSTM estándar entrenada en condiciones óptimas.

Sin embargo, aún queda mucho trabajo para explorar este campo. Sería interesante estudiar cómo influye la inyección de ruido en la función de activación en otras redes más complejas, como son la LSTM o la GRU. Otra alternativa sería combinar la idea de la arquitectura de la red Dual utilizando otros modelos de redes recurrentes, que posiblemente proporcionen interesantes resultados. Por último, una nueva alternativa podría consistir en estudiar una función de coste que considerase no solo el coste tradicional, como cross-entropy o el error cuadrático medio, sino que penalizara también la entropía de la activación de la capa recurrente, buscando modelos más sencillos sin necesidad de introducir ruido o regularización.

Bibliografía

[1] Z. Zeng, R. M. Goodman, and P. Smyth, "Learning finite state machines with self-clustering recurrent networks," *Neural Computation*, vol. 5, no. 6, pp. 976–990, 1993.

[2] M. Casey, "The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction," *Neural Computation*, vol. 8, no. 6, pp. 1135–1178, 1996.

[3] C. W. Omlin and C. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, vol. 9, no. 1, pp. 41 – 52, 1996.

[4] C. L. Giles, C. B. Miller, D. Chen, G. Z. Sun, H. H. Chen, and Y. C. Lee, "Extracting and learning an unknown grammar with recurrent neural networks," in *Advances in Neural Information Processing Systems 4* (J. E. Moody, S. J. Hanson, and R. P. Lippmann, eds.), pp. 317–324, Morgan-Kaufmann, 1992.

[5] M. Cohen, A. Caciularu, I. Rejwan, and J. Berant, "Inducing regular grammars using recurrent neural networks," *CoRR*, vol. abs/1710.10453, 2017.

[6] C. Oliva and L. F. Lago-Fernández, "Interpretability of recurrent neural networks trained on regular languages," in *Advances in Computational Intelligence - 15th International Work-Conference on Artificial Neural Networks, IWANN 2019, Gran Canaria, Spain, June 12-14, 2019, Proceedings, Part II* (I. Rojas, G. Joya, and A. Català, eds.), vol. 11507 of *Lecture Notes in Computer Science*, pp. 14–25, Springer, 2019.

[7] C. Oliva and L. F. Lago-Fernández, "On the interpretation of recurrent neural networks as finite state machines," in *Artificial Neural Networks and Machine Learning - ICANN 2019: Theoretical Neural Computation - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part I* (I. V. Tetko, V. Kurková, P. Karpov, and F. J. Theis, eds.), vol. 11727 of *Lecture Notes in Computer Science*, pp. 312–323, Springer, 2019.

[8] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[9] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. [van der Laak], B. [van Ginneken], and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60 – 88, 2017.

[10] T. Mikolov, M. Karafiát, L. Burget, J. ernocký, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH*, 2010.

[11] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[13] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179 – 211, 1990.

[14] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013.

[15] D. Eck and J. Schmidhuber, "Learning the long-term structure of the blues.," pp. 284–289, 01 2002.

[16] H. Siegelmann and E. Sontag, "On the computational power of neural nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132 – 150, 1995.

[17] H. T. Siegelmann, "Recurrent neural networks and finite automata," *Computational Intelligence*, vol. 12, no. 4, pp. 567–574, 1996.

[18] H. Jacobsson, "Rule extraction from recurrent neural networks: A taxonomy and review," *Neural Computation*, vol. 17, pp. 1223–1263, 06 2005.

[19] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014.

[20] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwinska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, pp. 471–476, 2016.

[21] J.-P. Bernardy, "Can recurrent neural networks learn nested recursion," 2018.

[22] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[23] C. Oliva and L. F. Lago-Fernández, "Dual recurrent neural network trained on spanish text generation," in *Artificial Neural Networks and Machine Learning - ICANN 2020: 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15-18, 2020. Submitted*, 2020.

[24] C. Oliva and L. F. Lago-Fernández, "Stability of internal states in recurrent neural networks trained on regular languages." `https://arxiv.org/abs/2006.10828`, 2020.

[25] C. Oliva and L. F. Lago-Fernández, "Separation of memory and processing in dual recurrent neural networks." `https://arxiv.org/abs/2005.13971`, 2020.

[26] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[27] N. Chomsky, "Three models for the description of language," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124, 1956.

[28] S. C. Kleene, "Representation of events in nerve nets and finite automata," in *Automata Studies* (C. Shannon and J. McCarthy, eds.), pp. 3–41, Princeton, NJ: Princeton University Press, 1956.

[29] P. Linz, *An Introduction to Formal Language and Automata*. USA: Jones and Bartlett Publishers, Inc., 2006.

[30] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies* (C. Shannon and J. McCarthy, eds.), pp. 129–153, Princeton, NJ: Princeton University Press, 1956.

[31] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal, The*,

vol. 34, pp. 1045–1079, Sept 1955.

[32] F. Karlsson, "Constraints on multiple center-embedding of clauses," *Journal of Linguistics*, vol. 43, pp. 365 – 392, 07 2007.

[33] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. 2, no. 42, pp. 230–265, 1936.

[34] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[35] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, *Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming*, pp. 151–170. Dordrecht: Springer Netherlands, 1996.

[36] D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*. 01 1998.

[37] M. Nielsen, "Neural networks and deep learning," 2015.

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[39] J. Schmidhuber, "Deep learning in neural networks: An overview," *CoRR*, vol. abs/1404.7828, 2014.

[40] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[41] C. L. Giles, G.-Z. Sun, H.-H. Chen, Y.-C. Lee, and D. Chen, "Higher order recurrent networks and grammatical inference," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 380–387, Morgan-Kaufmann, 1990.

[42] J. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, pp. 227–252, 01 1991.

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[44] A. Joulin and T. Mikolov, "Inferring algorithmic patterns with stack-augmented recurrent nets," *CoRR*, vol. abs/1503.01007, 2015.

[45] M. Collier and J. Beel, "Implementing neural turing machines," *CoRR*, vol. abs/1807.08518, 2018.

[46] F. A. Gers and E. Schmidhuber, "Lstm recurrent networks learn simple context-free and context-sensitive languages," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.

[47] A. Cleeremans, D. Servan-Schreiber, and J. Mcclelland, "Finite state automata and simple recurrent networks," *Neural Computation - NECO*, vol. 1, pp. 372–381, 09 1989.

[48] J. J. Michalenko, A. Shah, A. Verma, S. Chaudhuri, and A. B. Patel, "Finite automata can be linearly decoded from language-recognizing RNNs," in *International Conference on Learning Representations*, 2019.

[49] Q. Wang, K. Zhang, A. G. O. II, X. Xing, X. Liu, and C. L. Giles, "An empirical evaluation of recurrent neural network rule extraction," *CoRR*, vol. abs/1709.10380, 2017.

[50] J. Kolen, "Fool's gold: Extracting finite state machines from recurrent network dynamics.," pp. 501–

508, 01 1993.

[51] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[52] C. Bothe, C. Weber, S. Magg, and S. Wermter, "A context-based approach for dialogue act recognition using simple recurrent neural networks," *CoRR*, vol. abs/1805.06280, 2018.

[53] S. Chollampatt and H. Ng, "Neural quality estimation of grammatical error correction," pp. 2528–2539, 01 2018.

[54] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *CoRR*, vol. abs/1906.08237, 2019.

[55] G. Melis, T. Kočiský, and P. Blunsom, "Mogrifier lstm," 2019.

[56] D. Wang, C. Gong, and Q. Liu, "Improving neural language modeling via adversarial training," *CoRR*, vol. abs/1906.03805, 2019.

[57] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," *CoRR*, vol. abs/1708.02182, 2017.

[58] S. Takase, J. Suzuki, and M. Nagata, "Direct output connection for a high-rank language model," *CoRR*, vol. abs/1808.10143, 2018.

[59] Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen, "Breaking the softmax bottleneck: A high-rank RNN language model," *CoRR*, vol. abs/1711.03953, 2017.

[60] B. Krause, E. Kahembwe, I. Murray, and S. Renals, "Dynamic evaluation of neural sequence models," *CoRR*, vol. abs/1709.07432, 2017.

[61] G. Montavon, W. Samek, and K. Müller, "Methods for interpreting and understanding deep neural networks," *CoRR*, vol. abs/1706.07979, 2017.

[62] H. Strobelt, S. Gehrmann, B. Huber, H. Pfister, and A. Rush, "Visual analysis of hidden state dynamics in recurrent neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, 06 2016.

[63] A. Karpathy, J. Johnson, and F. Li, "Visualizing and understanding recurrent networks," *CoRR*, vol. abs/1506.02078, 2015.

[64] C. Needham, "Internet movie database (imdb)." `https://www.imdb.com/`, 1990.

[65] M. Tomita, "Dynamic construction of finite automata from examples using hill-climbing," in *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, (Ann Arbor, Michigan), pp. 105–108, 1982.

[66] K.-C. Jim, C. Giles, and W. Horne, "An analysis of noise in recurrent neural networks: Convergence and generalization," *Neural Networks, IEEE Transactions on*, vol. 7, pp. 1424 – 1438, 12 1996.

[67] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995.

[68] Ç. Gülçehre, M. Moczulski, M. Denil, and Y. Bengio, "Noisy activation functions," *CoRR*, vol. abs/1603.00391, 2016.

[69] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on*

Learning Representations, 12 2014.

Apéndices

A

Implementación de la Noisy RNN

Este apéndice contiene el código en lenguaje $Python$ de la implementación en *Keras* de la clase *NoisyRNN*. El código utiliza como base el código fuente de la clase *SimpleRNN* de *Keras*, por lo que solamente se muestran en el apéndice las funciones con modificaciones, indicando con un comentario dónde se encuentra la nueva implementación. El resto mantiene el mismo software que el código original.

A.1. Clase celda

Código A.1: Clase NoisyRNNCell - Constructor - Parte 1.

```
1  class NoisyRNNCell(Layer):
2
3      def __init__(self, units,
4                   activation='tanh',
5                   use_bias=True,
6                   kernel_initializer='glorot_uniform',
7                   recurrent_initializer='orthogonal',
8                   bias_initializer='zeros',
9                   kernel_regularizer=None,
10                  recurrent_regularizer=None,
11                  bias_regularizer=None,
12                  kernel_constraint=None,
13                  recurrent_constraint=None,
14                  bias_constraint=None,
15                  dropout=0.,
16                  recurrent_dropout=0.,
17                  noise_level=0., ##### NOISY RNN
18                  **kwargs):
19          super(NoisyRNNCell, self).__init__(**kwargs)
20          self.units = units
21          self.activation = activations.get(activation)
22          self.use_bias = use_bias
```

Código A.2: Clase NoisyRNNCell - Constructor - Parte 2.

```
23          self.kernel_initializer = initializers.get(kernel_initializer)
24          self.recurrent_initializer = initializers.get(recurrent_initializer)
25          self.bias_initializer = initializers.get(bias_initializer)
26
27          self.kernel_regularizer = regularizers.get(kernel_regularizer)
28          self.recurrent_regularizer = regularizers.get(recurrent_regularizer)
29          self.bias_regularizer = regularizers.get(bias_regularizer)
30
31          self.kernel_constraint = constraints.get(kernel_constraint)
32          self.recurrent_constraint = constraints.get(recurrent_constraint)
33          self.bias_constraint = constraints.get(bias_constraint)
34
35          self.dropout = min(1., max(0., dropout))
36          self.recurrent_dropout = min(1., max(0., recurrent_dropout))
37          self.state_size = self.units
38          self.output_size = self.units
39          self._dropout_mask = None
40          self._recurrent_dropout_mask = None
41
42          self.noise_level = noise_level ##### NOISY RNN
```

Código A.3: Clase NoisyRNNCell - Función call - Parte 1.

```
1       def call(self, inputs, states, training=None):
2           prev_output = states[0]
3
4           if 0 < self.dropout < 1 and self._dropout_mask is None:
5               self._dropout_mask = _generate_dropout_mask(
6                   K.ones_like(inputs),
7                   self.dropout,
8                   training=training)
9           if (0 < self.recurrent_dropout < 1 and
10                  self._recurrent_dropout_mask is None):
11              self._recurrent_dropout_mask = _generate_dropout_mask(
12                  K.ones_like(prev_output),
13                  self.recurrent_dropout,
14                  training=training)
15
16          dp_mask = self._dropout_mask
17          rec_dp_mask = self._recurrent_dropout_mask
18
19          if dp_mask is not None:
20              h = K.dot(inputs *dp_mask, self.kernel)
```

Código A.4: Clase NoisyRNNCell - Función call - Parte 2.

```
21          else:
22              h = K.dot(inputs, self.kernel)
23          if self.bias is not None:
24              h = K.bias_add(h, self.bias)
25
26          if rec_dp_mask is not None:
27              prev_output *= rec_dp_mask
28          output = h + K.dot(prev_output, self.recurrent_kernel)
29
30          ################################################
31          # NOISY RNN
32          # Introducimos el ruido aqui.
33          ################################################
34          noise = self.noise_level*prev_output*K.random_normal(prev_output.shape)
35          output += noise
36          ################################################
37
38          if self.activation is not None:
39              output = self.activation(output)
40
41          # Properly set learning phase on output tensor.
42          if 0 < self.dropout + self.recurrent_dropout:
43              if training is None:
44                  output._uses_learning_phase = True
45          return output, [output]
```

A.2. Clase NoisyRNN

Código A.5: Clase NoisyRNN - Constructor - Parte 1.

```
class NoisyRNN(RNN):
    @interfaces.legacy_recurrent_support
    def __init__(self, units,
                 activation='tanh',
                 use_bias=True,
                 kernel_initializer='glorot_uniform',
                 recurrent_initializer='orthogonal',
                 bias_initializer='zeros',
                 kernel_regularizer=None,
                 recurrent_regularizer=None,
                 bias_regularizer=None,
                 activity_regularizer=None,
                 kernel_constraint=None,
                 recurrent_constraint=None,
                 bias_constraint=None,
                 dropout=0.,
                 recurrent_dropout=0.,
                 noise_level=0., ##### NOISY RNN
                 return_sequences=False,
                 return_state=False,
                 go_backwards=False,
                 stateful=False,
                 unroll=False,
                 **kwargs):
        if 'implementation' in kwargs:
            kwargs.pop('implementation')
            warnings.warn('The `implementation` argument '
                          'in `SimpleRNN` has been deprecated. '
                          'Please remove it from your layer call.')
        if K.backend() == 'theano' and (dropout or recurrent_dropout):
            warnings.warn(
                'RNN dropout is no longer supported with the Theano backend '
                'due to technical limitations. '
                'You can either set `dropout` and `recurrent_dropout` to 0, '
                'or use the TensorFlow backend.')
            dropout = 0.
```

Código A.6: Clase NoisyRNN - Constructor - Parte 2.

```
37              recurrent_dropout = 0.
38
39          cell = NoisyRNNCell(units,
40                              activation=activation,
41                              use_bias=use_bias,
42                              kernel_initializer=kernel_initializer,
43                              recurrent_initializer=recurrent_initializer,
44                              bias_initializer=bias_initializer,
45                              kernel_regularizer=kernel_regularizer,
46                              recurrent_regularizer=recurrent_regularizer,
47                              bias_regularizer=bias_regularizer,
48                              kernel_constraint=kernel_constraint,
49                              recurrent_constraint=recurrent_constraint,
50                              bias_constraint=bias_constraint,
51                              dropout=dropout,
52                              recurrent_dropout=recurrent_dropout,
53                              noise_level=noise_level ##### NOISY RNN
54                              )
55          super(NoisyRNN, self).__init__(cell,
56                                         return_sequences=return_sequences,
57                                         return_state=return_state,
58                                         go_backwards=go_backwards,
59                                         stateful=stateful,
60                                         unroll=unroll,
61                                         **kwargs)
62          self.activity_regularizer = regularizers.get(activity_regularizer)
```

B

Estados de la red: Tomita 3

Este apéndice contiene todos los resultados de las transiciones del espacio interno de estados de la red de Elman modificada cuando es entrenada con ruido para aprender el lenguaje *Tomita 3*. Con estas figuras se pretende ilustrar que la red de Elman modificada genera una serie de clusters estables con transiciones completamente definidas y, en todos los casos, también deterministas, que pueden interpretarse como el conjunto de estados de un autómata finito determinista.

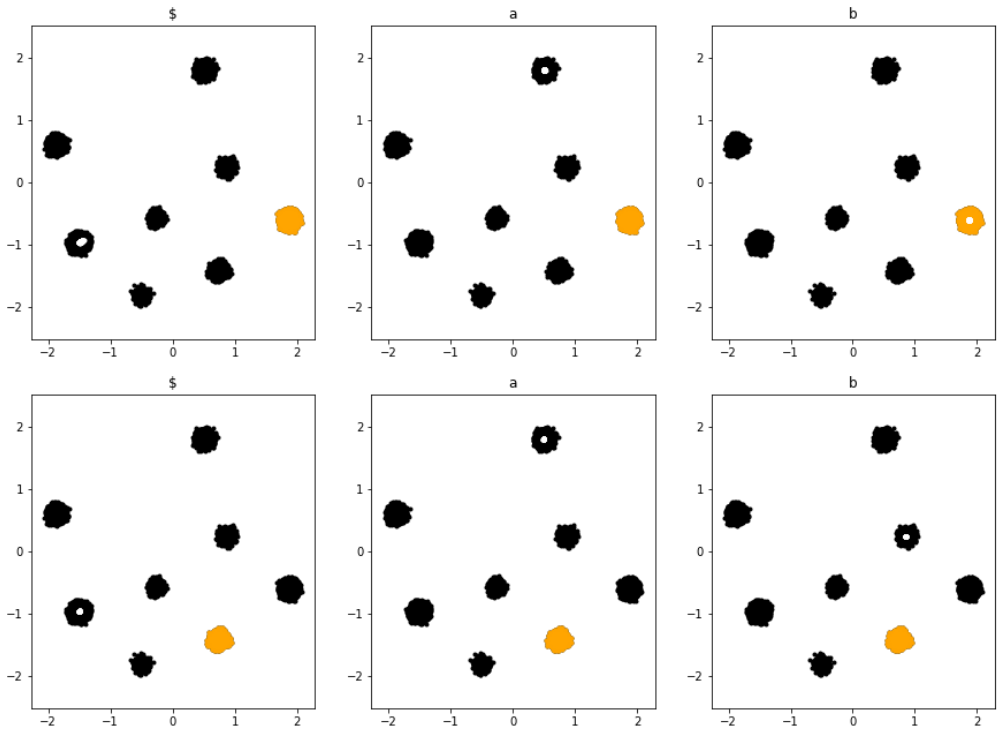


Figura B.1: Transiciones de los estados coloreados en naranja con cada uno de los símbolos del alfabeto para resolver el problema *Tomita 3*. Los puntos blancos en el interior de los clusters representan los destinos reales de cada uno de los puntos. Nótese cómo en todos los casos se aproximan al centro del cluster.

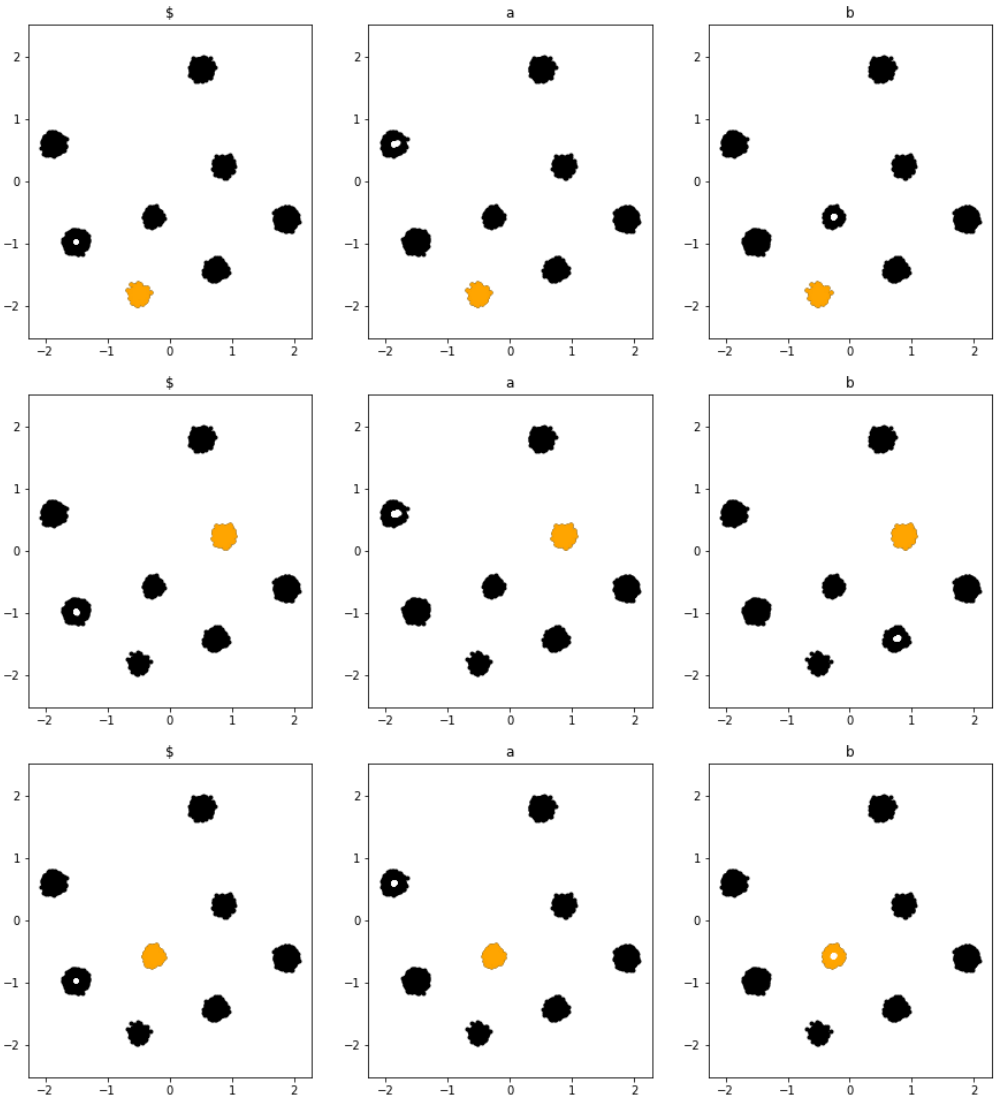


Figura B.2: Transiciones de los estados coloreados en naranja con cada uno de los símbolos del alfabeto para resolver el problema *Tomita 3*. Los puntos blancos en el interior de los clusters representan los destinos reales de cada uno de los puntos. Nótese cómo en todos los casos se aproximan al centro del cluster.

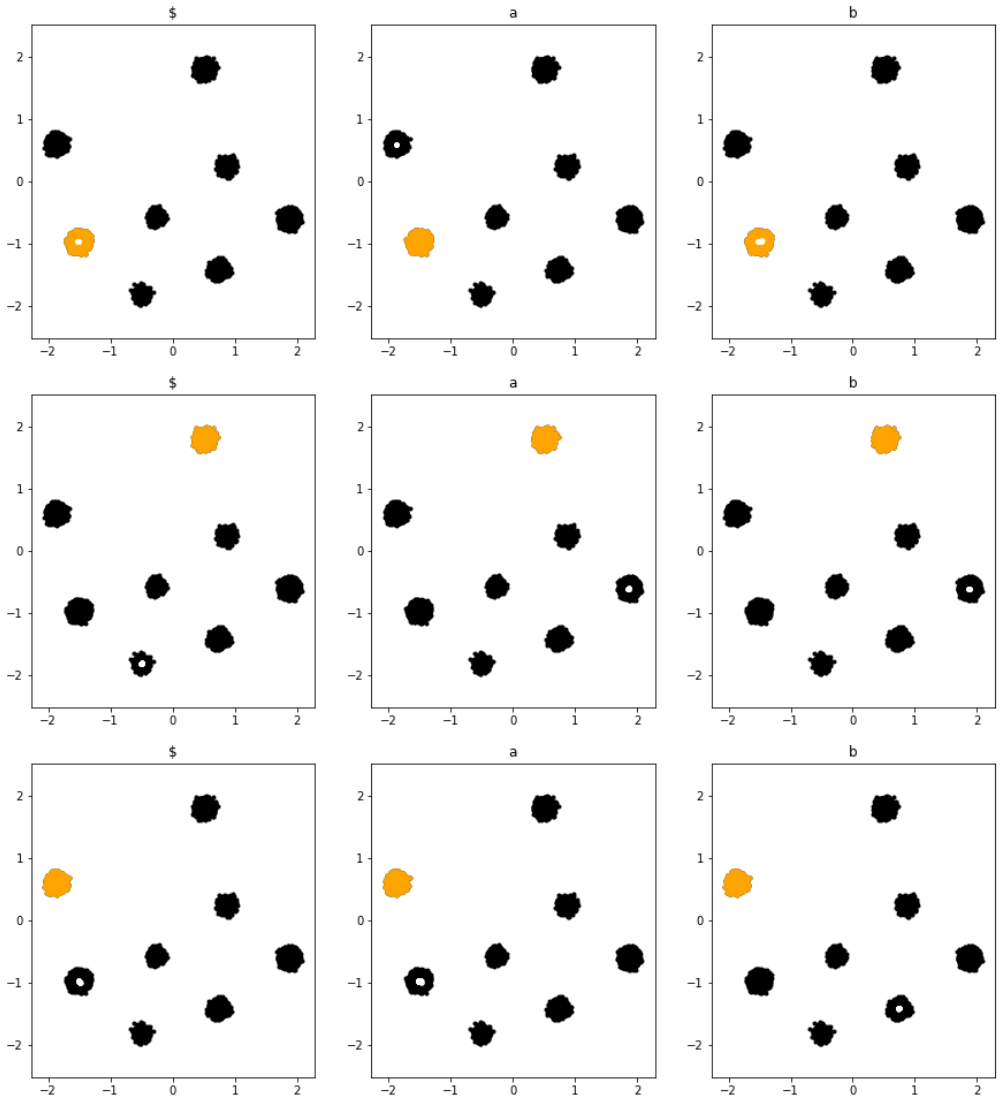


Figura B.3: Transiciones de los estados coloreados en naranja con cada uno de los símbolos del alfabeto para resolver el problema *Tomita 3*. Los puntos blancos en el interior de los clusters representan los destinos reales de cada uno de los puntos. Nótese cómo en todos los casos se aproximan al centro del cluster.

C

Extracción del autómata: Tomitas

En este apéndice se muestran las tablas de transiciones entre estados de la red de Elman entrenada con cada una de las *Tomita Grammars* [65]. Con estos resultados se demuestra que es posible extraer el autómata finito determinista mínimo que resuelve cada uno de los problemas a partir de la extracción del autómata y su correspondiente minimización.

	a	b	$
0*	0	1	0
1	1	1	0

		a	b	$
→	0* (0)	0	1	0
	1 (1)	1	1	0

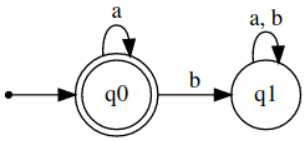


Figura C.1: Tabla de transiciones de los clusters formados para resolver el problema *Tomita 1* y el autómata finito determinista extraído.

	a	b	$
0*	2	1	0
1	0	2	0
2	2	2	0

		a	b	$
→	0* (0)	2	1	0
	1 (1)	0	2	0
	2 (2)	2	2	0

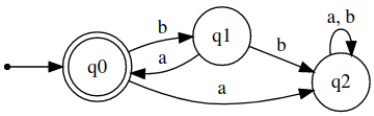


Figura C.2: Tabla de transiciones de los clusters formados para resolver el problema *Tomita 2* y el autómata finito determinista extraído.

	a	b	$
0*	7	5	3
1	6	1	3
2*	7	2	2
3*	7	4	2
4*	7	4	2
5	6	0	2
6	1	1	3
7*	3	5	3

		a	b	$
	0* (1, 6)	0	0	4
	1 (5)	0	2	4
	2* (0)	3	1	4
	3* (7)	4	1	4
→	4* (2, 3, 4)	3	4	4

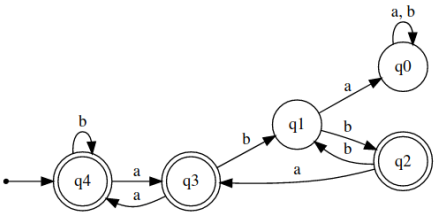


Figura C.3: Tabla de transiciones de los clusters formados para resolver el problema *Tomita 3* y el autómata finito determinista extraído.

	a	b	$
0*	1	4	0
1*	3	2	0
2*	3	0	3
3*	3	2	0
4	4	4	3

		a	b	$
0*	(0)	1	4	1
1*	(1, 3)	1	2	1
2*	(2)	1	0	1
4	(4)	4	4	3

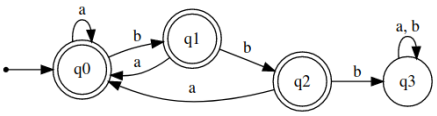


Figura C.4: Tabla de transiciones de los clusters formados para resolver el problema *Tomita 4* y el autómata finito determinista extraído.

	a	b	$
0*	5	4	0
1	4	3	0
2*	5	4	2
3	2	1	2
4	1	0	0
5	2	1	0

			a	b	$
→	0*	(0, 2)	2	3	0
	1	(1)	3	2	0
	2	(3, 5)	0	1	0
	3	(4)	1	0	0

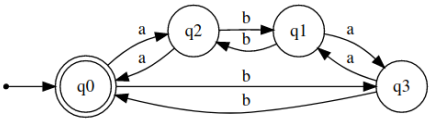


Figura C.5: Tabla de transiciones de los clusters formados para resolver el problema *Tomita 5* y el autómata finito determinista extraído.

	a	b	$
0	6	1	4
1*	3	2	4
2	4	0	5
3	6	1	5
4*	3	2	4
5*	3	2	4
6	4	0	4

			a	b	$
	0	(0, 3)	2	1	1
→	1*	(1, 4, 5)	0	2	1
	2	(2, 6)	1	0	1

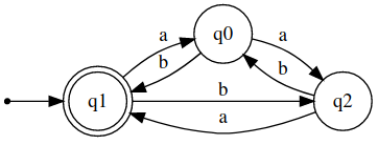


Figura C.6: Tabla de transiciones de los clusters formados para resolver el problema *Tomita 6* y el autómata finito determinista extraído.

	a	b	$
0*	2	0	0
1*	2	1	1
2*	2	3	0
3*	4	3	0
4*	4	5	0
5	5	5	1

			a	b	$
→	0*	(0, 1)	1	0	0
	1*	(2)	1	2	0
	2*	(3)	3	2	0
	3*	(4)	3	4	0
	4	(5)	4	4	0

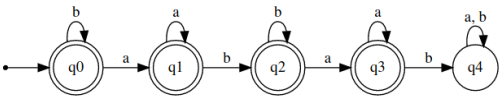


Figura C.7: Tabla de transiciones de los clusters formados para resolver el problema *Tomita 7* y el autómata finito determinista extraído.

D

Generalización y estabilidad

En este apéndice se muestran los resultados de la ejecución de la prueba con el conjunto de datos *mega* para estudiar la estabilidad de la red de Elman con ruido y compararla con una LSTM estándar. Como se puede observar en las figuras, mientras que la red de Elman con ruido mantiene el 100 % de acierto, la LSTM tiene una brusca caída en el acierto de algunos problemas.

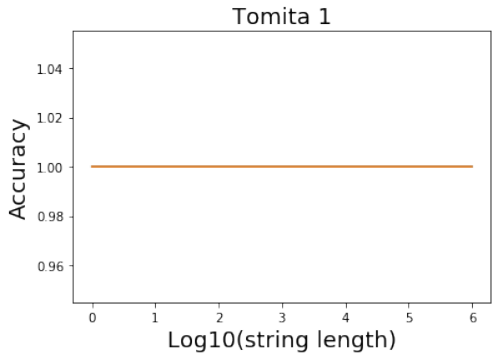


Figura D.1: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Tomita 1* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

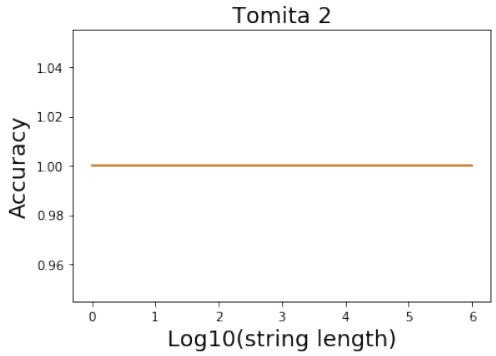


Figura D.2: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Tomita 2* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

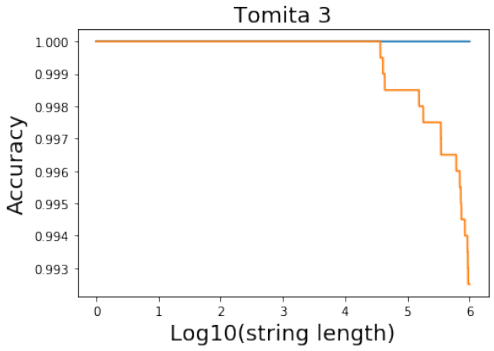


Figura D.3: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Tomita 3* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

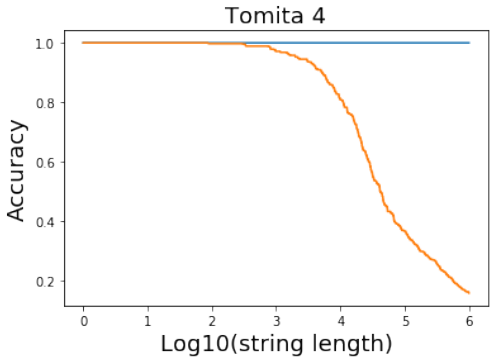


Figura D.4: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Tomita 4* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

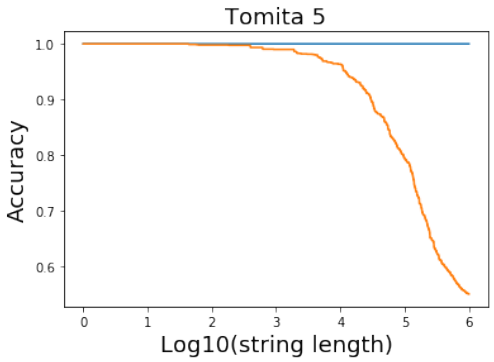


Figura D.5: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Tomita 5* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

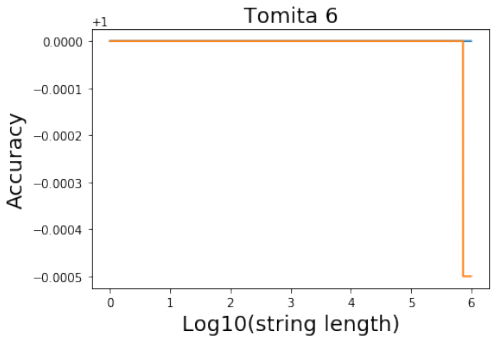


Figura D.6: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Tomita 6* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

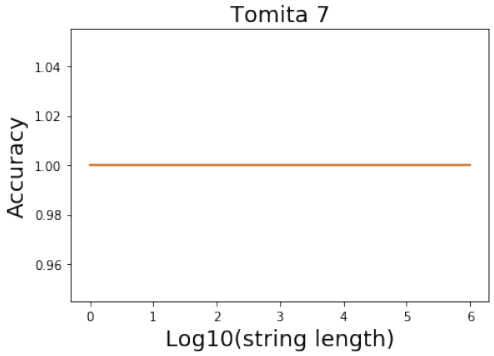


Figura D.7: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Tomita 7* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

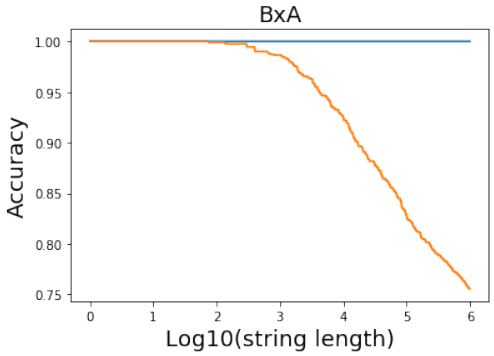


Figura D.8: Acierto frente a la longitud de cadena en escala logarítmica para el problema *BxA* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.

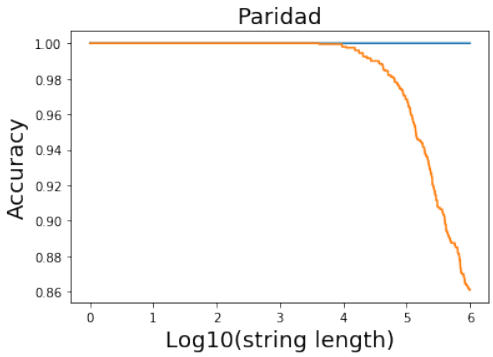


Figura D.9: Acierto frente a la longitud de cadena en escala logarítmica para el problema *Paridad* con una Noisy Elman RNN ($\nu = 1{,}0$) y una LSTM estándar. Nótese la escala logarítmica en el eje horizontal.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID